C5/6xxx Hardware Reference Guide

Includes

- SigC5xxx and SigC6xxx Boards
- M6x Boards
- DSK C5xxx and C6xxx Boards
- DSP, CCS, and Host Software Support

Copyright ã 2002-2006 Signalogic, Inc.
Revision G1
Mar 2006

Signalogic, Inc.
9617 Wendell @ Skillman
Dallas, TX 75243
Tel: 214-349-5551
Fax: 214-343-0163

http://www.signalogic.com
tech_support@signalogic.com

TABLE OF CONTENTS

1 System Overview 8
1.1 Documentation Notes and Conventions 9
1.2 Additional Documentation Sources 9
1.3 Hardware 10
1.4 Software 11
2 Software Installation 13
2.1 Installing Code Composer Studio (CCS) Software 15
2.1.1 Removing Previous Versions of CCS 15
2.1.2 Creating a Merged Installation of CCS v2.x 16
2.1.3 Important Install Steps 17
2.1.4 Installing a JTAG Emulator 18
2.1.5 Installing JTAG Emulator Software 19
2.1.5.1 Testing the JTAG Emulator 23
2.1.6 Configuring CCS to Run with the JTAG Emulator 28
2.1.7 Making Sure Processors are Not in Reset 33
2.1.8 Running CCS 33
2.2 Installing Signalogic Software 34
2.2.1 Electronic Format Software Upgrade 34
2.2.2 Electronic Format Software Upgrade for MELPe 35
2.2.3 Directory Structures 35
2.2.4 WinXP Driver Install Instructions 39
2.2.4.1.1 Installing PC/104 and PTMC Board WinXP Drivers 39
2.2.4.1.2 Installing M6x Board WinXP Drivers 43
2.2.5 Linux Driver Install Instructions 53
2.2.5.1 SigC5xxx and SigC6xxx PC104 Linux Driver Install Instructions 53
2.2.5.2 SigC641x and SigC5561 PTMC Linux Driver Install Instructions 53
2.2.6 Hypersignal-Macro Software Board Support 54
3 Hardware Installation 56
3.1 Installing SigC5xxx and SigC6xxx PC/104 Hardware 57
3.1.1 Installing PC/104 Boards in a PC 57
3.1.2 Installing PC/104 Boards in a Stand-Alone Stack 57
3.1.3 PC104 Board Installation Steps 58
3.1.4 Additional Notes For Stand-Alone PC/104 Board Systems 65
3.1.5 Power Consumption Reduction PC104 Board Modifications 65
3.2 Installing SigC55xx and SigC64xx PTMC Hardware 66
3.2.1 Installing PTMC Cards in a PC 66
3.2.2 Installing PTMC Cards on a cPCI Carrier Board 68

3.2.3 PTMC Card Installation Steps 69
3.3 Installing M6x Hardware 74
3.4 Installing Texas Instruments DSK C54xx and DSK C67xx Boards 75
3.5 Hardware Test and Check-Out Programs 78
3.5.1 Stand-Alone Mode Test 78
3.5.1.1 CCS Startup and Stand-Alone Mode Test 79
3.5.1.2 CCS Startup and Stand-Alone Mode Test for MELPe 83
3.5.1.3 Flash EEPROM Boot and Stand-Alone Mode Test 84
3.5.2 Board Peripheral Testing 84
3.5.2.1 Basic DSP HPI Testing 84
3.5.2.2 DMA Controller Testing 85
3.5.2.3 Flash EEPROM Testing 85
3.5.3 Linux Test Programs 86
3.5.3.1 Running Board and Memory Test Programs 86
3.5.3.2 Running Data Scope Test Program 87
3.5.3.3 Running the E1-Audio Test Program 89
3.5.3.4 Running the DUART Test Program 96
3.5.3.5 Running PTMC Card Memory Test Programs 98
4 Building DSP Application Software 102
4.1 CCS Project Build & Run Sequence 103
4.1.1 SCI Builds 114
4.1.2 DSP/BIOS Builds 114
4.1.3 MINCONFIGx Active Configurations 114
4.1.4 Talker – Host Interface Driver 116
4.1.5 Cache – Enabling and Organizing Code 116
4.1.6 Application Code Performance and Architecture Considerations 119
4.1.7 CCS Build Output, File Types, and File Locations 120
4.1.8 COFF File Format 121
4.2 Host Mode vs. Stand-Alone Mode 122
4.3 Incorporating Peripheral I/O Access into Applications 125
4.4 DSP Code Memory Map Considerations 127
4.5 DSP Boot Code 129
4.6 Integrating Off-the-Shelf Algorithms 130
5 Building Host Application Software 141
5.1 Building Visual Studio Software 142
5.1.1 Running Visual C/C++ and Visual Basic Programs with DSP Hardware 142
5.1.2 Interface Between Visual C/C++ and Real-Time DSP Code 147
5.2 Building MATLAB Software 153

5.2.1 Running MATLAB Programs with DSP Hardware 153
5.2.2 Interface Between MATLAB and Real-Time DSP Code 155
5.3 Building Linux Software 163
5.3.1 Rebuilding boardTest and dataScope Test Programs 163
5.3.2 Rebuilding the E1-Audio Test Program 163
5.3.3 Rebuilding the DUART Test Program 164
5.3.4 Rebuilding LKM Drivers 164
5.3.4.1 Rebuilding SigC5xxx and SigC6xxx PC104 LKM Drivers 164
5.3.4.2 Rebuilding the SigC641x and SigC5561 PTMC LKM Drivers 165
5.3.5 Application Layer Source Code 166
6 Developing and Debugging DSP Code 169
6.1 Host Mode and UserProc Debugging 171
6.1.1 Hypersignal-Macro Simple FFT Function 171
6.1.2 UserProc: Adding C Code to Real-Time DSP Processing 177
6.1.3 Probing Real-Time C Code from Hypersignal 183
6.1.4 Probing Real-Time C Code from DirectDSP-Enabled Visual C/C++ and Visual Basic Programs 184
6.1.5 Probing Real-Time C Code from MATLAB Programs 184
6.2 Stand-Alone Mode Debugging 185
6.3 Code Composer Studio Debugging Examples 186
6.4 CCS JTAG Issues During Debug 187
Hardware Issues During Debug 193
7 DSP Software API 194
7.1 Peripheral Access Functions 194
7.1.1 Digital I/O Functions 195
7.1.2 DMA Controller Functions 199
7.1.3 DUART Functions 200
7.1.4 Flash EEPROM Functions 201
7.1.5 Host Bus Functions 202
7.1.6 LED Functions 203
7.1.7 Routing Logic Functions 204
8 TBD 208
9 Module Interfaces 209
9.1 MiniPCI Module Interfaces 211
9.2 SODIMM 72 Module Interfaces 212
9.3 Module Host Processor Interface 214
10 Board Resources and Peripherals 218

10.1 Host Interface 219
10.1.1 PC104 Host Interface 220
10.1.2 PCI Host Interface 222
10.1.2.1 PCI Configuration Space Values 224
10.1.2.2 BAR0 Description 224
10.1.2.3 BAR1 Description 231
10.1.2.4 BAR2 Description 232
10.2 DSP HPI Register Access 233
10.3 Logic Registers 235
10.4 Peripheral Registers 241
10.5 DMA Controller 245
10.5.1 DMA Controller Testing 245
10.5.2 Initial DMA Configuration 245
10.5.3 DMA Controller Performance 245
10.6 DUART 247
10.6.1 DUART Testing 247
10.6.2 DUART Performance Notes 247
10.7 Flash/Boot EEPROM 248
10.7.1 Flash EEPROM Testing 248
10.7.2 EEPROM Programming Instructions 248
10.7.2.1 Creating a ROM File 248
10.7.2.2 Programming Flash EEPROM 249
10.7.2.3 Configuring the Board for Stand-Alone Operation 251
10.7.3 EEPROM Boot Flow 252
10.7.4 EEPROM Storage Format 253
10.7.5 Executable DSP Program Storage Format in EEPROM 256
10.7.6 EEPROM Boot Program Listings 257
10.7.6.1 C5xxx EEPROM Boot Program Listing 257
10.7.6.2 C6xxx EEPROM Boot Program Listing 258
10.8 Digital I/O 260
10.8.1 Digital I/O Testing 262
10.8.2 Digital I/O Hardware Implementation Notes 263
10.8.3 Digital I/O Example C Code 263
10.9 Routing Logic 266
10.9.1 Routing Logic Example C Code 270
10.10 T1/E1 273
10.11 RS-422 274
10.12 USB 2.0 276

10.13 Ethernet 802.3 277

10.14 Ethernet 802.11b 278

11 Component-Level Test and Measurement 279

11.1 Measuring Board Power Consumption 279

 Copyright © Signalogic 2003-2005

Revision History

Created	Rev A	Mar 2003	Consolidated from several individual documents
Revised	Rev B	Jul 2003	
Revised	Rev C	Sep 2003	
Revised	Rev D	Nov 2003	
Revised	Rev E	Jan 2004	
Revised	RevE1-E4	Feb 2004	
Revised	Rev E5	Mar 2004	
Revised	Rev E6-E9	Jun 2004	
Revised	Rev F	Jul 2005	Added SigC64xx board sections.
Revised	Rev G	Sep 2005	Updated SigC64xx board sections.

 Copyright © Signalogic 2003-2005

1 System Overview

This document assumes you are using a Pentium-compatible or PowerPC-based system, running Win9x, Win2K, WinXP, or Linux, and you will install and operate the SigC5xxx and SigC6xxx PC/104 or PTMC boards, M6x board, or Texas Instruments DSK C6xxx and DSK C54xx boards.

In order to run the software and programs described in this guide, either a standard PC (desktop, lab machine, or laptop), stand-alone PC/104 board stack, or cPCI carrier board should be available and configured as shown in the following sections.

1.1 Documentation Notes and Conventions

Below are some common conventions used in this documentation.

1) Command line examples in the following sections beginning with

```
shell/>
```

indicate a command that needs to be typed in at the shell command prompt.

2) “WinXP” refers also to Win2K systems.

3) The words “board” and “card” are used interchangeably to refer to DSP based electronic circuit boards. Generally, the word card is used when there is a possibility of confusing the DSP card with a motherboard or carrier board. In that case, the word card is used in the sense of “daughtercard”.

4) The nomenclature

```
..\
```

is used to refer to software installation subfolders in some cases. For example, to refer to .obj files in shorthand and include both C5xxx and C6xxx hardware, the documentation might say “.obj files are located on ..\SCI\cStandard subfolders” which means .obj files can be found on

```
\dspcode\c5xxx\SCI\cStandard
```

subfolder for C5xxx hardware, and on

```
\dspcode\c6xxx\SCI\cStandard
```

subfolder for C6xxx hardware.

5) HSM and HSA indicate Hypersignal-Macro and Hypersignal-Acoustic software (Signalogic).

6) CCS indicates Code Composer Studio software (Texas Instruments).

1.2 Additional Documentation Sources

This manual is located online at:

ftp://ftp.signalogic.com/documentation/Boards/SigC55xx

and at

ftp://ftp.signalogic.com/documentation/Boards/SigC64xx

 Copyright © Signalogic 2003-2005

1.3 Hardware

Some possible board and hardware combinations that you may have are given below.

SigC5/6xxx PC104 Boards

- SigC5xxx-PC/104 board or SigC6xxx-PC/104, Rev. C1 or C2
- SigC5416-SODIMM module, Rev. B or v1.3x
- SigC5409A-SODIMM module, Rev B or v1.3x
- SigC6711-SODIMM module, Rev. A or B
- SigSD4 Audio Module, Rev. A or B (optional, depending on application requirements)

- **Optional** Pentium-compatible main PC/104 board, for example standard desktop system using ISA or PCI passive carrier card, or embedded PC/104 system such as Lippert CoolRunner II or Diamond Systems

SigC5/6xxx PTMC Cards

- SigC6415-PTMC board, v1.0
- SigC5561-PTMC board, v1.0

- **Optional** HW400cxx cPCI carrier board and wanT3 PTMC daughtercard; for example "blade" configuration that combines PowerPC, T3 interface, Ethernet interface and DSP/VoIP resources in one cPCI slot

M6x Boards

- M62 or M67 PCI Board
- A4D4, SD4, SD16, A16D16, AIX, AIX20, or DAC40 module(s)

Note that M6x boards are full-sized PCI boards that plug into a host (typically desktop) PC slot.

Texas Instruments DSK Boards

- DSK C6711
- DSK C6211
- DSK C5402

Note that DSK boards are external "lab development boards" that do not have a bus connector and are intended to connect to a PC via USB or parallel port.

Optional Digital I/O board or other third-party boards

 Copyright © Signalogic 2003-2005

1.4 Software

Below is a summary of software tools referred to in this documentation.

Software	Provider	Layer	Overview	Comments
Hypersignal-Macro (HSM) or Hypersignal-Acoustic (HSA)	Signalogic	Application	Utilities, Diagnostic, Test, Measurement and Real-Time Instrument Display, basic DSP and Math functions	Optional, depends on application
DirectDSP software Win9x and WinXP version, including • Hardware Library DLL, v3.1 • Engine Manager DLL, v3.1 • Hardware Manager DLL, v3.1	Signalogic	Library / API	DSP Hardware Interface	Win9x and WinXP
DirectDSP software Linux version [1], including • Hardware Library shared object module, v1.2 • Engine Manager shared object module, v1.2 • Hardware Manager shared object module, v1.2	Signalogic	Library / API	DSP Hardware Interface	Linux only
"VxD" style virtual device driver	Signalogic	Driver	Low-level hardware interface	Win9x systems only [2]
Kernel mode drivers	Signalogic	Driver	Low-level hardware interface	.sys and .inf files for Win2k and WinXP systems Loadable kernel module (LKM) driver, v1.2 for Linux [1]
C5xxx or C6xxx Source Code Interface (SCI)	Signalogic	Library / API	"Board Support Package", includes DSP Source and Library Code, Example Projects	Optional, depends on application
Code Composer Studio (CCS) [3]	Texas Instruments	Application	DSP Code Development	Optional, depends on application

Visual Studio	Microsoft	Application	Host Code Development	Typically used to build GUI and host-based applications that will make DirectDSP API calls to interface to DSP hardware. Optional, depends on application
MATLAB	The MathWorks	Application	Typically used with DirectDSP interface. Optional, depends on application	DSP algorithm design and simulation; can use hardware interface for data acquisition, DSP algorithm coding test, fixed-point precision measurement, etc Optional, depends on application

Notes

[1] Assumes Red Hat Linux v7.3 or higher, Kernel v2.4.18-3 or higher.

[2] Win95, Win98, or Win98 2nd Edition.

[3] CCS software is developed by Texas Instruments and available from a variety of sources, including Signalogic. CCS software is an IDE with compiler, assembler, and linker tools for modifying existing or developing new C5xxx and C6xxx DSP code. There are two (2) versions of CCS software: "full version" with JTAG emulation and debug capability, and "DSK version" without JTAG debug. JTAG debug refers to low-level debug functions such as breakpoint, single-step, and register access.

 Copyright © Signalogic 2003-2005

2 Software Installation

Sections 2.1 and 2.2 below cover installing Code Compose Studio (CCS) software and Signalogic software.

Also, please read the following "Important Software Install Notes" section, as they contain some highlights and crucial points to observe for a successful and smooth installation.

Important Software Install Notes

1) You *must* successfully install CCS and JTAG emulator software first before taking any other step. Signalogic software can be installed after basic stand-alone mode hardware testing is first performed (see section 3.5.1.1 below, CCS Startup and Stand-Alone Mode Test).

Signalogic software will allow far greater hardware test, as well as measurement and analysis, advanced DSP real-time access and data transfer, algorithm level debug, advanced GUI, and other capability not provided by CCS, but the minimum required software for any type of board testing is CCS software.

2) Locate either on the Spectrum Digital CD or download from SD's website the latest JTAG emulator drivers and copy to your \ti\drivers subdir. The files to locate are:

sdgo5xx.dvr	C5xxx hardware
sdgo6x.dvr	C6xxx hardware

Note that the Spectrum Digital JTAG emulator drivers will work with other emulators covered in this document, including SEED.

3) Run CCS "Setup Utility" – this is critical. The Setup Utility will prompt you with an installation wizard series of dialog boxes that allow you to configure CCS for the correct driver (listed above), the number of C5xxx or C6xxx processors on your hardware, and make other key CCS/driver settings, such as base I/O address for the SD driver (typically 0x378).

4) After completing the CCS Setup Utility installation wizard, the utility will prompt you to run CCS on exit. ***Answer yes and do this! It is absolutely crucial to do this.*** Whatever the Setup Utility does at this point is mysterious, but a good thing.

5) After finishing with the CCS Setup Utility, you should be able to run the SDConfig program successfully on the SigC5xxx or SigC6xxx PC/104 board (see section 3.5.1.2 below).

6) For PC/104 hardware, make sure jumper JP14 is installed before running CCS in stand-alone mode. JP14 indicates stand-alone mode boot, and is required for correct CCS stand-alone operation.

7) For MELPe demo code or other demo programs that utilize RS-422, make sure you have the RS-422 "loopback plug" installed on J30 connector (see board pictures in section 3.1 below).

 Copyright © Signalogic 2003-2005

This plug connects pins 2 and 3 and allows digital algorithm data (e.g. for MELPe, compressed speech data) to be looped back in the default demo code.

2.1 Installing Code Composer Studio (CCS) Software

Installation of Texas Instruments Code Composer Studio (CCS) software is described below. Note that version v2.1 or higher of (CCS) is assumed.

2.1.1 Removing Previous Versions of CCS

Although strictly speaking not necessary, it has been a historical issue with CCS software that upgrades and new versions can run into problems when a previous version is installed. Texas Instruments has improved this issue over the years, but for some reason it continues to persist in some cases. Therefore, it is a safe precaution to first clean your PC system of any existing TI tools, or any CCS version for any of the DSP series, but leave your data files.

Before proceeding, PLEASE READ THIS WARNING FROM TEXAS INSTRUMENTS:

> *Errors in the registry can prevent your PC from booting properly. You should make a backup of your registry just in case something is done incorrectly that causes problems with your system. If you are not comfortable modifying the registry, contact your computer support group for assistance.) This information is provided as a service from Texas Instruments. However, Texas Instruments assumes NO liability for problems caused by incorrectly modifying the registry.*

Please follow these steps to completely remove CCS from your machine.

- Go through the Add/Remove programs step in Control panel to remove Code Composer Studio.

- When the un-installation is complete, check to see if there is a "details" button to the left of the "OK" button. If a "details" button does exist, please click on it and look at the directories that are remaining and then delete them from your drive.

- Next, go to the Start menu and select Run. Type “regedit” and click Ok (this will cause the Registry Editor window to appear). Next, go to the following places in the registry and delete the Go DSP, TI and Texas Instruments keys (if they continue to exist).

```
HKEY_CURRENT_USER -> Software -> Go DSP

HKEY_CURRENT_USER -> Software -> Texas Instruments

HKEY_CURRENT_USER -> Software -> TI
```

- In the registry now go to HKEY_LOCAL_MACHINE -> SOFTWARE. From here delete the GO DSP and TI and Texas Instruments keys (if they exist).

```
HKEY_LOCAL_MACHINE -> Software -> Go DSP
```

```
HKEY_LOCAL_MACHINE -> Software -> Texas Instruments

HKEY_LOCAL_MACHINE -> Software -> TI
```

After following the steps above, reboot your machine. By eliminating all of the references to Code Composer Studio from your computer, you will be installing Code Composer Studio on a clean machine.

- Must be the full version with all needed drivers. Please note that the CCS version offered on TI's web site (60-day trial version) will not work since it does not include all necessary drivers needed to work with a JTAG emulator. Also note that a "DSK version" of CCS (a copy of CCS supplied with a DSP Starter Kit board) will ***not work*** with a JTAG emulator; instead, all "DSK versions" of CCS assume they can perform JTAG emulation through either the PC parallel port or USB port.

NOTE 1: If you have any Signalogic programs installed on your system, you MUST take the following steps to ensure they continue to run correctly after the CCS installation.

1. Click the Start button.
2. Choose "Run".
3. Type "sysedit" in the pop-up window
4. The System Configuration Editor program will load, with several windows already open inside.
5. Click the "system.ini" window.
6. Scroll through the contents of the system.ini file until you reach the section of the file titled "[386enh]".
7. Below the "[386enh]" title, add the line "device = pipe.vxd"or ensure this line is already there. (This line may have been deleted during CCS installation)
8. Save the system.ini file, and close the System Configuration Editor.

NOTE 2: If you are installing only one version of CCS v2.x (either the C5000 or C6000 version), or are installing a CCS other than version v2.x, skip to section 2.1.4 below, Installing a JTAG Emulator.

2.1.2 Creating a Merged Installation of CCS v2.x

It is possible to install both the C5000 and C6000 versions of CCS v2.x to create a "C5000/C6000" installation of CCS v2.x. This unified version allows development for all C5000 and C6000 series processors on one machine. The following instructions assume creation of a unified installation.

 Copyright © Signalogic 2003-2005

2.1.3 Important Install Steps

Insert the CD for the other processor series version of Code Composer Studio. Choose “Install Code Composer Studio” from the menu. After you accept the licensing agreement, the program will offer two options. Choose the “Merge, Upgrade, or Reinstall an Existing Installation” option. If the CD in the drive is a separate version of CCS v2.x than the one already installed, this option will turn the previous installation into a combined installation.

2.1.4 Installing a JTAG Emulator

To install the Spectrum Digital XDS510PP, XDS510PP-Plus, or SEED XDSpp JTAG emulators, perform the following steps:

- Connect the emulator to a wall outlet with the provided ac-dc adapter. Note that in particular with the SEED emulator, there is a very specific adapter that comes with the emulator and you should always use this one

- Connect the emulator to the parallel port on the PC using the provided parallel port cables. *In your PC BIOS, the parallel port mode must be set to EPP mode – ECP-only mode will not work.*

- Connect the emulator's JTAG connector to the target board's JTAG header, and power-up the board ***– first, see section 3.1 below, Installing SigC5xxx and SigC6xxx PC/104 Hardware!***

NOTE 1: JTAG headers are keyed!! Pin 6 should be keyed – be careful to correctly orient the JTAG connector from the emulator with your target board header.

NOTE 2: If the board is connected to a PCI or ISA slot, power-down the computer before connecting the emulator.

The SEED emulator has the nice additional feature that a red LED will illuminate indicating that the emulator is plugged into the target board correctly. The Spectrum Digital emulator does not have this feature; instead it has a ridiculous little "Active" LED that is supposed to turn on when there is JTAG activity -- but basically you will never see this because you will be focused on your work inside CCS when JTAG activity is occurring.

An important note regarding length of JTAG cable: you will notice that for all emulators the 14-pin ribbon cable between the JTAG emulator enclosure and the target board header is very short. And in the case of the Spectrum Digital emulator, the cable is also surrounded by a grounded shield sleeve. There are very good reasons for such precautions. JTAG signals, especially the clock (TCK signal), are notoriously sensitive to noise and ringing. Especially, any degradation in clock edge rise-time will cause problems with TCK. It has to do with the fact the JTAG standard is now more than 15 years old and when first specified it used a non-differential level, clock-edge and data sampling scheme not suitable to rates more than 1 or 2 MHz. Since that time clock rates have, of course, increased dramatically to keep pace with larger DSP applications and more complex debug strategies. The minimum rate for a JTAG emulator now is 10 MHz, and some run up to 40 MHz. Therefore, it is always a good idea to minimize JTAG cable length – and if you are in the process of board design, that thinking extends to keeping the JTAG header on the target board very near the first DSP in the chain. As well, it is a good idea to buffer the TCK signal on the board – immediately after the emulator connector -- to ensure clean clock edges and fast rise-time.

 Copyright © Signalogic 2003-2005

2.1.5 Installing JTAG Emulator Software

First, download the drivers for either the Spectrum Digital XDS510PP, XDS510PP-PLUS or SEED XDSpp JTAG emulators to be used with CCS software. The drivers can be found on Spectrum Digital's website at:

http://www.spectrumdigital.com/ftp.asp

NOTE 1: There are different drivers for: a) different versions of CCS, and b) different processor families, so be sure to choose the correct drivers for your system.

NOTE 2: The Spectrum Digital drivers will work with the SEED emulator also.

Second, run the downloaded .exe file and follow all installation steps. For added simplicity, be sure to use the default directory Texas Instruments subdir for drivers also. The default CCS subdir is normally:

```
\ti
```

After installing the JTAG emulator, next install the Spectrum Digital Emulator Configuration Utility (called SDConfig). Like the drivers, SDConfig can be downloaded from:

http://www.spectrumdigital.com/ftp.asp

After downloading the SDConfig utility, click on the icon to run the utility (the icon should be on your desktop; if not you can create a shortcut from a typical subdir c:\specdig\sdconfg). You should see a display similar to Figure 2-1 below:

 Copyright © Signalogic 2003-2005

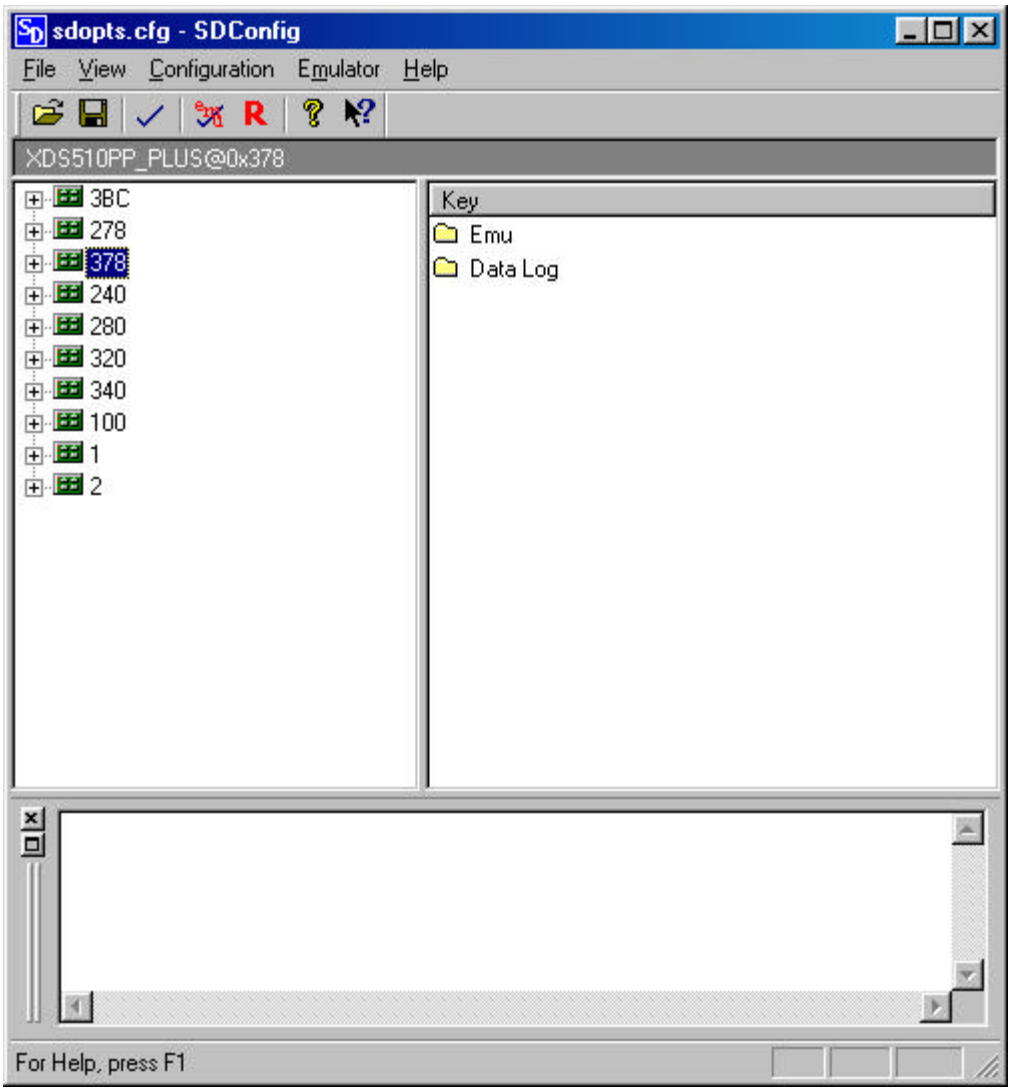


Figure 2-1, "SDConfig" Emulator Configuration Utility

Choose the icon representing the port to which the emulator is connected (normally this is LPT1, the PC's main parallel port, located at 0x378) and in the right-hand window, double-click the "Emu" subfolder. You should see a screen display similar to Figure 2-2 below:

 Copyright © Signalogic 2003-2005

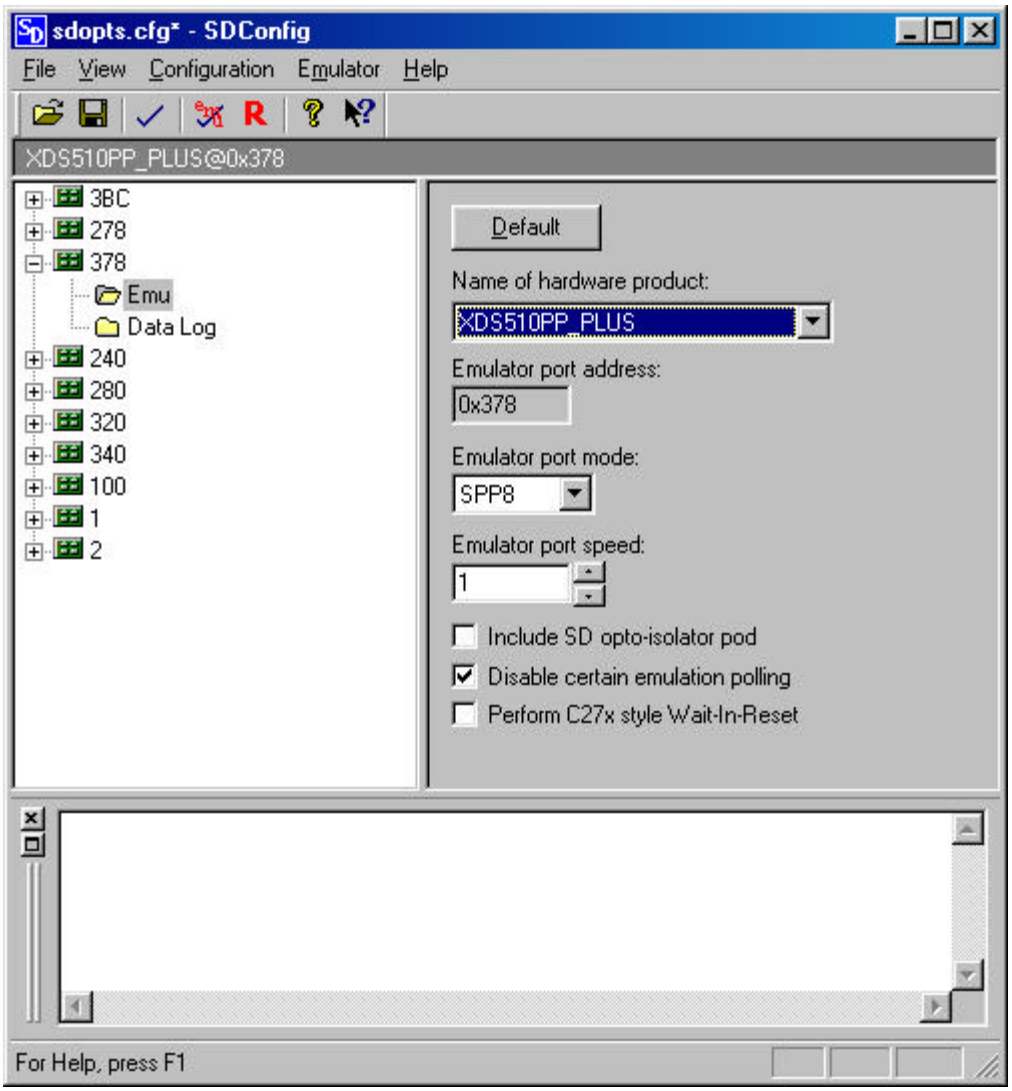


Figure 2-2, SDConfig Utility Configuration Display

First, select your emulator type from the drop-down menu, labeled “Name of hardware product”.

<u>NOTE 3</u>: is very important at this point to pay careful attention to whether or not you have selected XDS510PP or an XDS510PP Plus). Your selection <u>must</u> match the emulator type.

Below is a table with emulator type and correct SDConfig product selection.

Emulator Type	SDConfig Product Selection
Spectrum Digital XDS510PP	XDS510PP
Spectrum Digital XDS510PP-Plus	XDS510PP-Plus
SEED XDSpp	XDS510PP

Below is an example configuration display for a SEED XDSpp emulator installation:

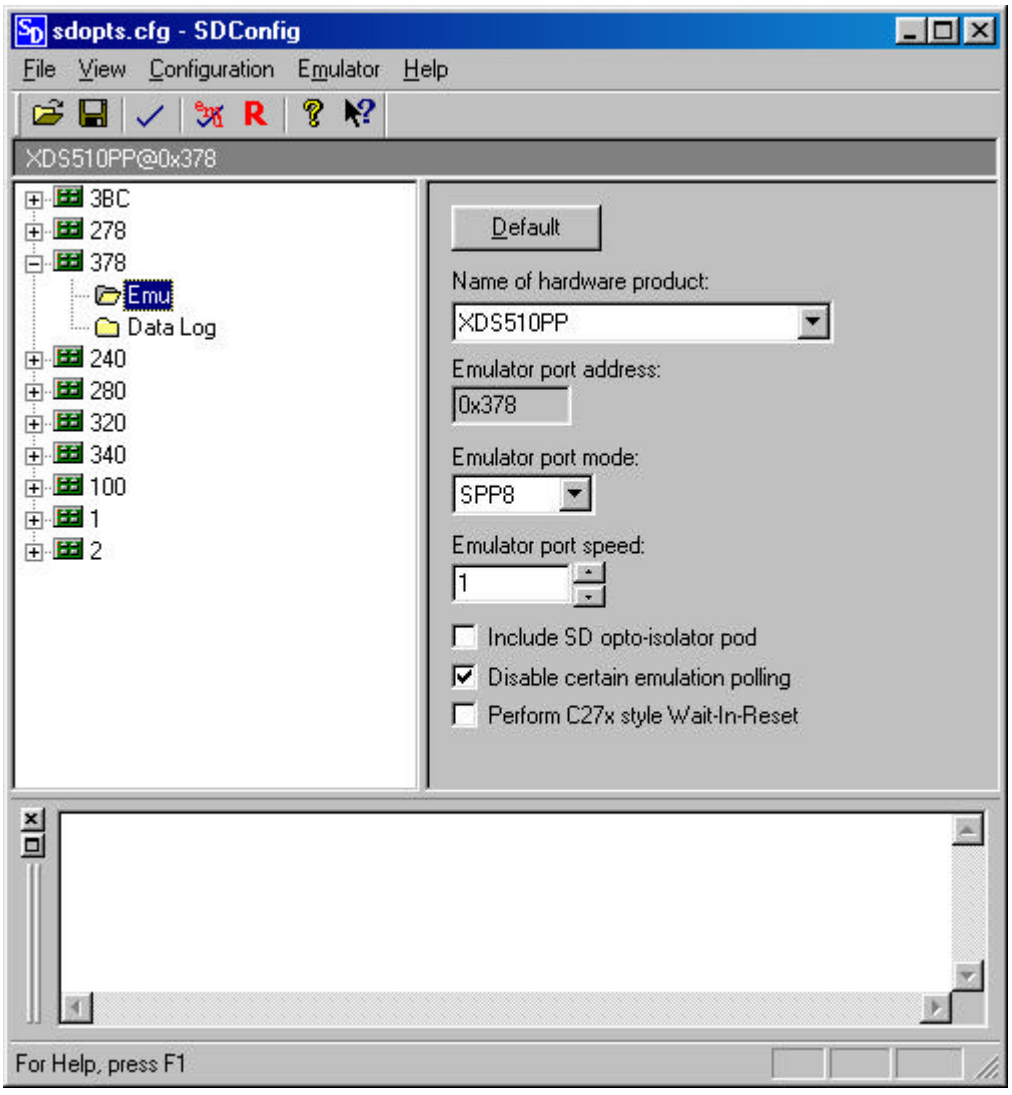


Figure 2-3, SDConfig Utility Configuration Display for SEED XDSpp Emulator

Next, choose the emulator port mode. SPP8 should work for all possible port modes. If SPP8 does not work, check this document for information on determining your port address and mode using Spectrum Digital's port I/O utility:

http://www.spectrumdigital.com/technical/pdfs/setup_chek.pdf

You can also select EPP mode. However, SDConfig does not seem to be consistent in saving EPP mode selection, although we have found this does not make any difference in emulator performance as the CCS software and emulator drivers always assume EPP mode.

***NOTE 4*: In any case, it is very important that you have EPP selected in your PC BIOS parallel port configuration. EPP v1.9, ECP+EPP, or other combination should be Ok. But without EPP, it is unlikely the emulator will work.**

For remainder of the setup files, the defaults will work in most cases. For detailed information on all fields in the setup area at right of the SDConfig display, look here:

ftp://ftp.signalogic.com/tech_support/CCS/SDConfig/SDConfig.htm

***NOTE 5*: Be sure to save your config file when exiting SDConfig ! You will find that you often run SDConfig to invoke the emulator Reset and Emu test toolbar buttons. This occurs frequently due to hang-ups in CCS due both to user-define code crashing (yep, you never have any bugs in your code, right) and also due to CCS strangenesses. Therefore it is important to always re-enter SDConfig with the proper setup information already saved.**

2.1.5.1 Testing the JTAG Emulator

After configuration, to test the JTAG emulator, first Reset the emulator by pressing the SDConfig toolbar button with a large red "R". You should see a message at the bottom of the SDConfig display window

```
"**Emulator is reset."
```

If you do not see the above message, verify your hardware connections and try again. Once the emulator has been reset, run the emulator test by clicking the button with an "emu" in front of a check mark. After running the emu command, for C54xx devices you should see:

```
Found JTAG IR length of 8
Found 1 JTAG device(s) in the scan chain
```

and for C67xx devices you should see:

```
Found JTAG IR length of 46
Found 1 JTAG device(s) in the scan chain
```

Below are some examples of SDConfig screen captures for different hardware types:

 Copyright © Signalogic 2003-2005

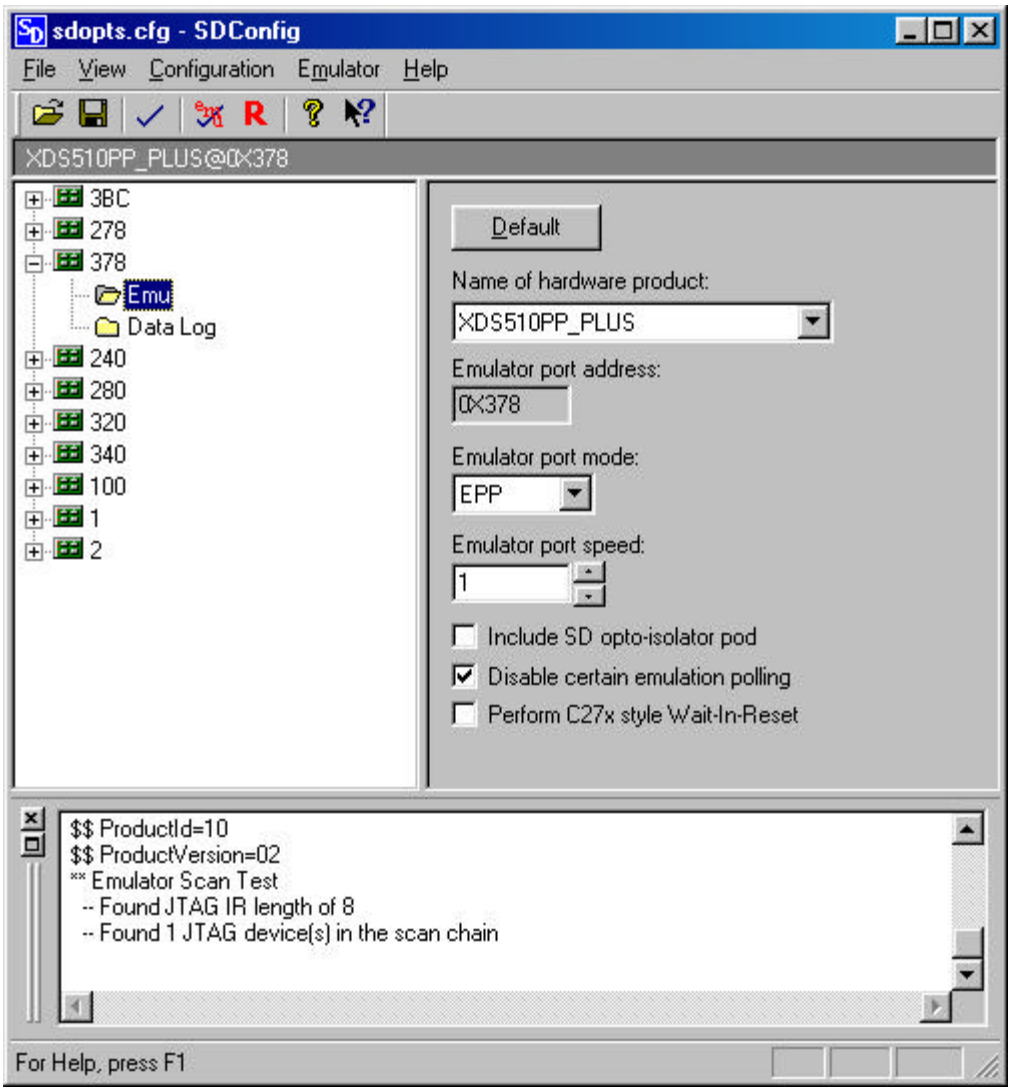


Figure 2-4, Example SDConfig Display for Single-Processor SigC54xx Board

 Copyright © Signalogic 2003-2005

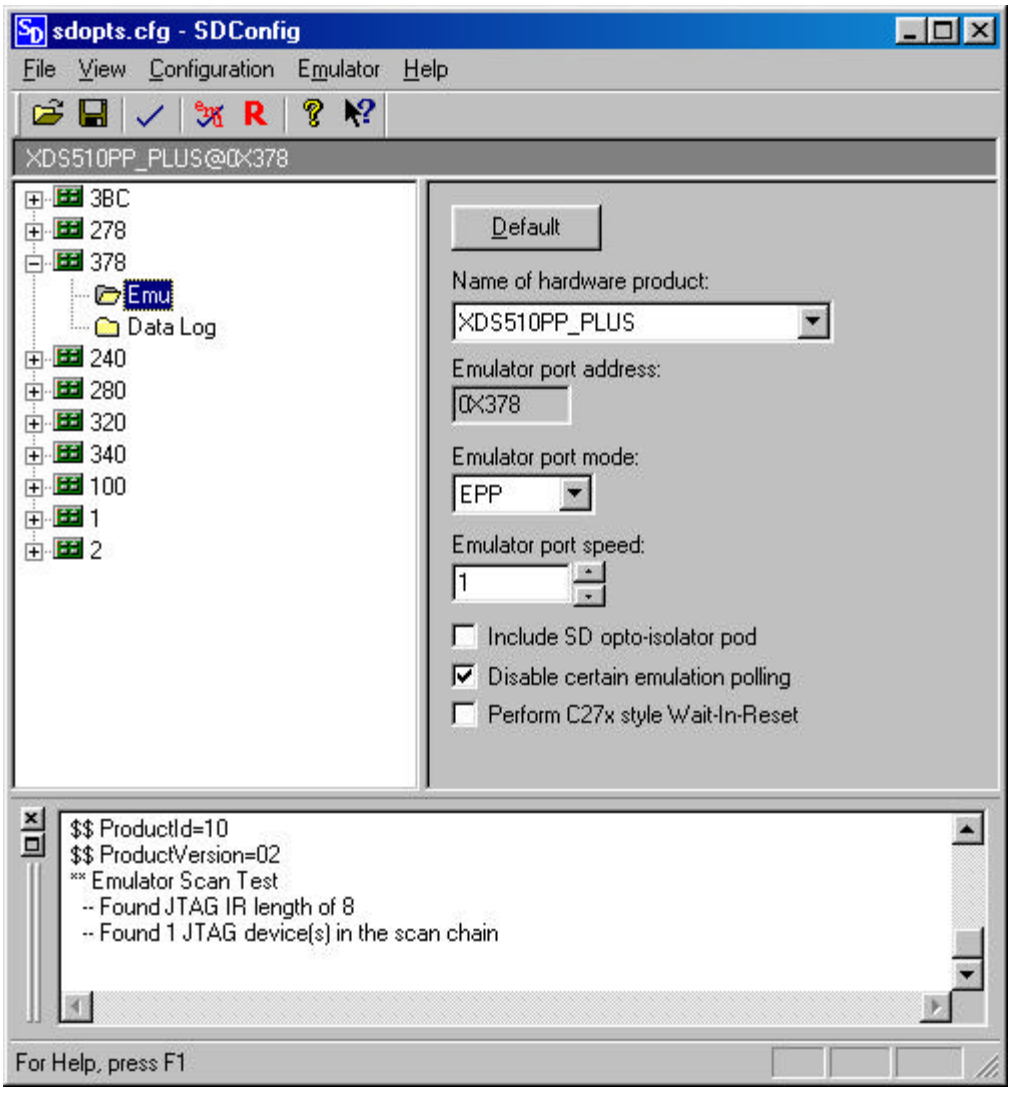


Figure 2-5, Example SDConfig Display for Single-Processor SigC67xx Board

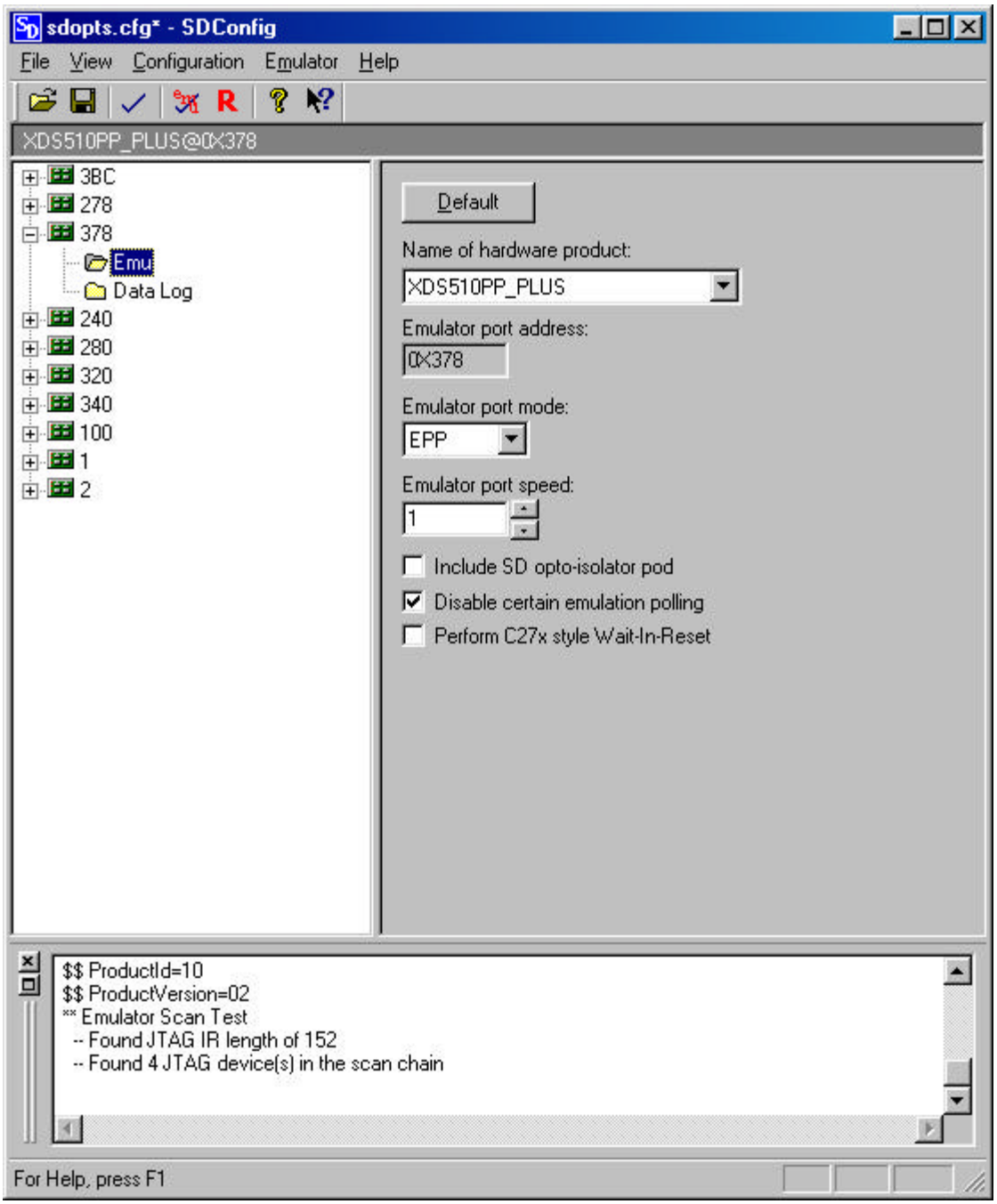


Figure 2-6, Example SDConfig Display for 4-Processor SigC64xx Board

Note that for multi-processor modules, the IR length and device values above should be multiplied by the number of DSP devices on the module.

If the test passes, CCS setup and configuration can proceed as described in the following sections. If either the reset or emulator test failed, please look at the links listed above for troubleshooting information. Also, below is a link that includes answers to frequently asked questions for XDS510PP-type emulators:

 Copyright © Signalogic 2003-2005

http://www.spectrumdigital.com/technical/pdfs/faq.pdf

2.1.6 Configuring CCS to Run with the JTAG Emulator

Click on the icon to run the setup program installed with CCS, which is typically called CCS Setup or CC_Setup in desktop icons, or on Texas Instruments entries under your Programs menu (the typical filename is cc_setup.exe).

After running CCS Setup, you should see a display similar to Figure 2-7 below:

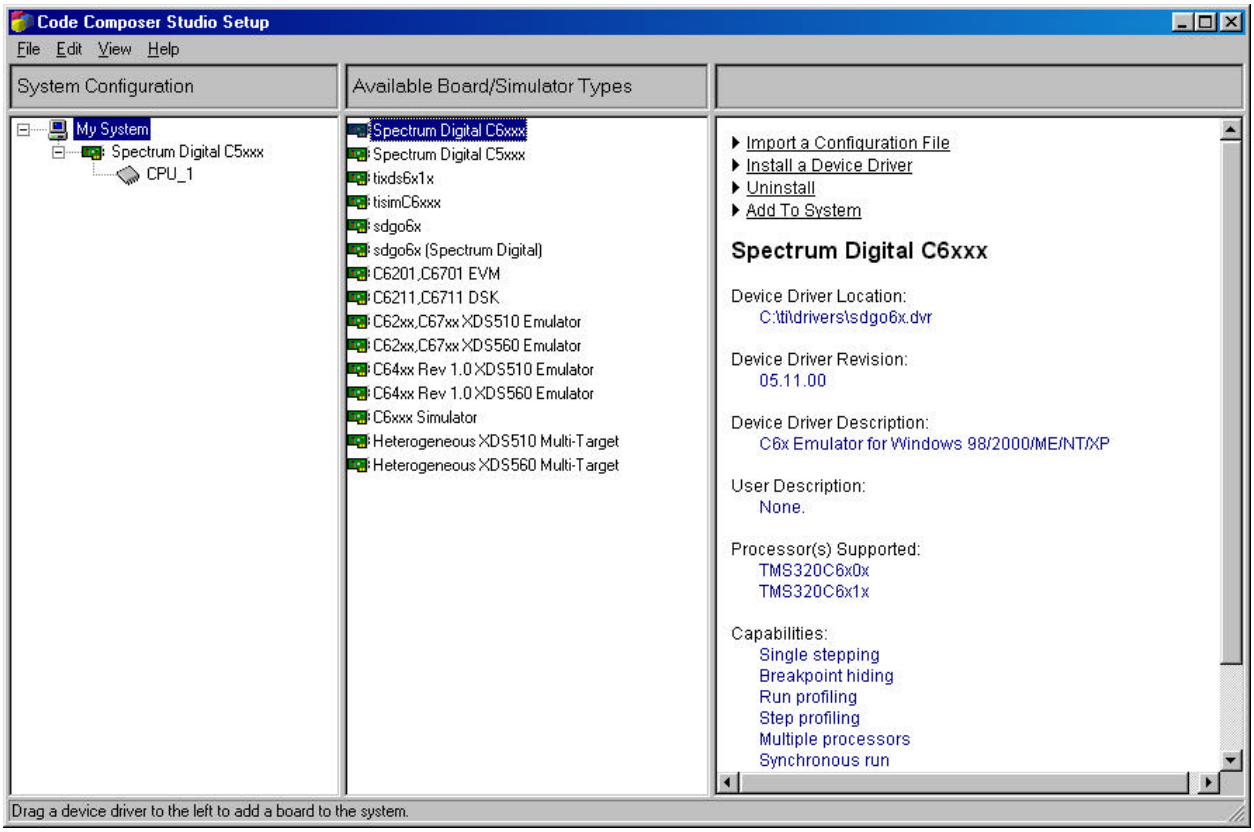


Figure 2-7, CCS Setup Main Display

The example display above was taken from a Signalogic lab and shows a number of drivers in the system. Regardless of the number of drivers in your system, you should first clear any existing configuration once you enter CCS Setup; i.e. there should be no green board icons in the left-most panel.

Next, from the right-most panel, select “Install a Device Driver.” The file selection box will open to the \ti\c5400\drivers subdirectory (or \ti\c6000\drivers subdirectory or as appropriate for the CCS installation). Select the file named “sdgoXXXX.dvr” (where XXXX is representative of the processor family to which the target board’s processor belongs). CCS Setup will then ask you to name this device configuration. After the configuration is named, new board icon will appear in the middle panel (with whatever name you assigned it).

Next, right-click on the new board icon, and select Properties from the popup menu. You should see a "Board Properties" wizard-style dialog appear, similar to Figure 2-8 below:

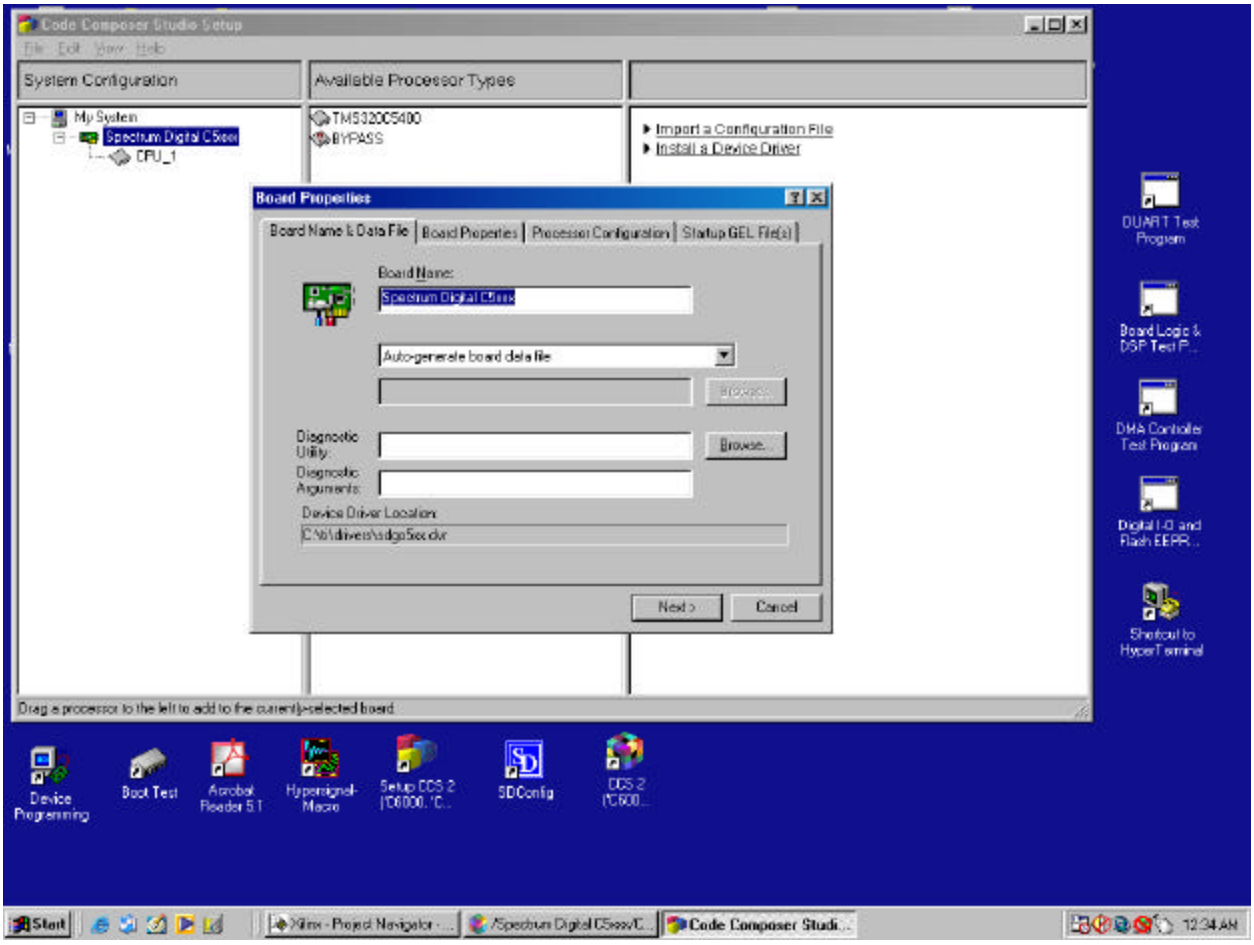


Figure 2-8, CCS Setup "Board Properties" Wizard-Style Dialog

Click the Next button to continue. You should see a display similar to Figure 2-9 below:

 Copyright © Signalogic 2003-2005

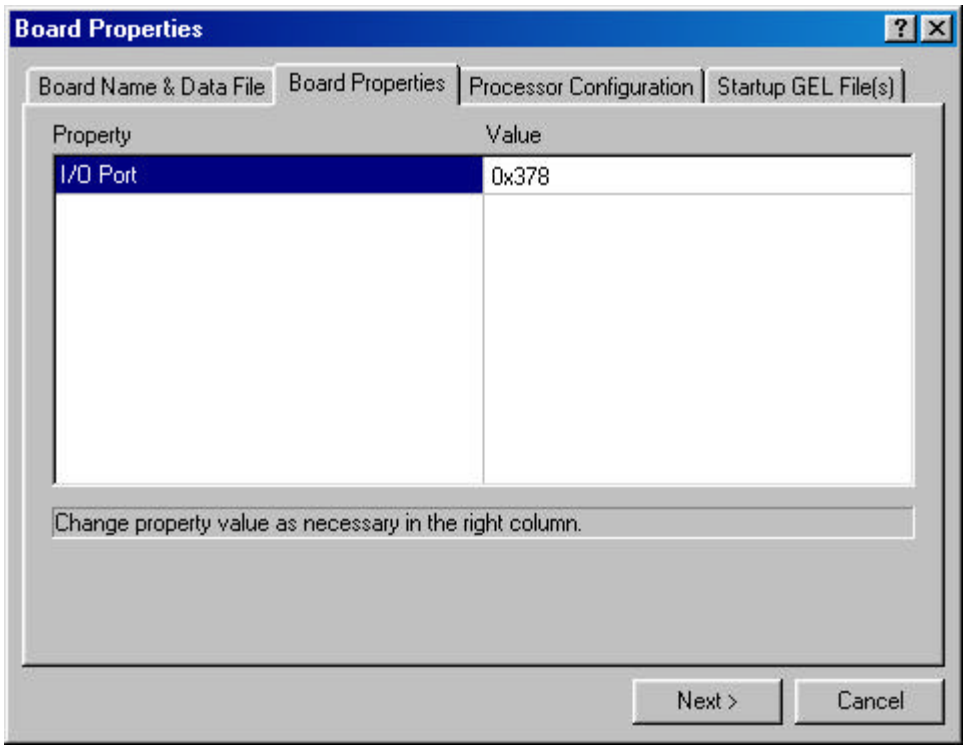


Figure 2-9, CCS Setup "Board Properties" Dialog Tab

The Board Properties tab will ask for the JTAG emulator I/O port. ***The I/O port entered here MUST match the port to which the emulator is connected! This should match your previous setup in the SDConfig utility (see section 2.15 above).***

After the I/O port is entered, click Next. You should see a display similar to Figure 2-10 below:

 Copyright © Signalogic 2003-2005

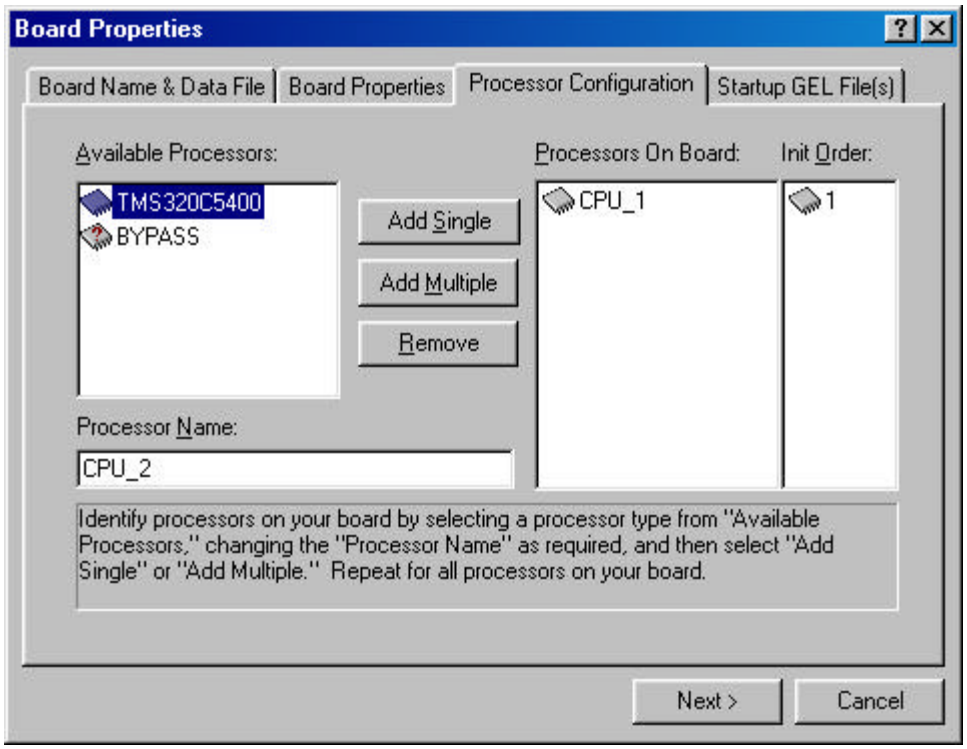


Figure 2-10, CCS Setup "Processor Configurations" Dialog Tab

Add processors to the configuration by highlighting them and selecting "Add Single" or "Add Multiple." If you have more than one processor in your JTAG chain, be sure to click on Add Multiple the correct number of times. Obviously, it is important to choose the processor types that correspond to the processors on the board.

NOTE 1: If there are multiple processors on the board of different types, the processors must be added to the configuration in the order they are laid out on the JTAG scan chain.

After entering processor configuration, click Next again, and a display similar to Figure 2-11 below should appear:

 Copyright © Signalogic 2003-2005

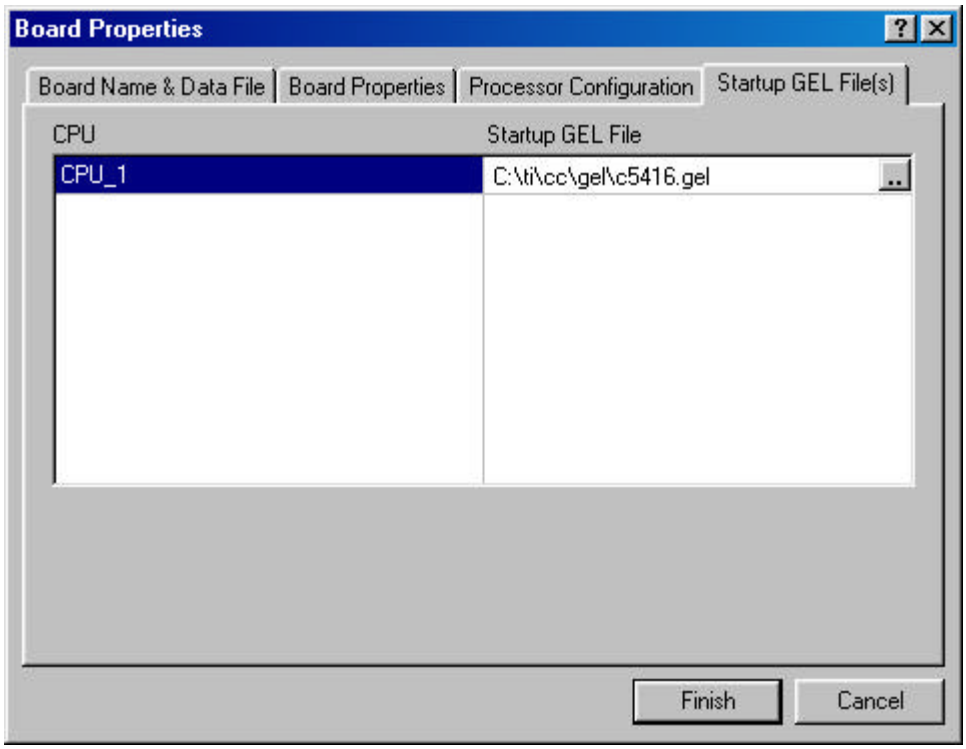


Figure 2-11, CCS Setup "Startup GEL File" Dialog Tab

In general, you should specify a GEL file here that matches your processor type. GEL files are used by CCS to initialize basic, crucial board settings, such as memory map, wait-states, processor configuration (e.g. microcontroller mode vs. microprocessor mode), etc. Without the correct GEL file, CCS can exhibit erratic and strange behavior upon start-up and when downloading and running executable DSP.

In some cases there are specific reasons to not load a GEL file, for example if you are testing low-level boot code or simulating a situation where the processor is booting from Flash EEPROM and the target hardware is responsible for making all initial DSP settings (for example, if a separate microcontroller or logic controller on the board is handling this).

If Signalogic has provided a processor-specific or board-specific GEL file with the Source Code Interface software, then this would override the default TI selection, and in that case make sure to add it to the "Startup GEL File(s)" tab.

Upon clicking "Finish", the board configuration and processors added will be in the left-most panel in the main CCS Setup (see Figure 2-7 above). From the File menu, choose "Save." CCS is now ready to use with the JTAG emulator.

NOTE 2: When complete with CCS Setup, it will prompt you whether to "Run CCS on Exit". Always answer yes to this question, even if only to close or abort CCS immediately afterwards. There is something mysterious about what CCS Setup does at that point, but whatever that it is, if you try to skip it, you can have problems later.

***NOTE 3*: If you have any problem in running CCS at this point, for example if it does not initialize correctly, then it is good idea to run the SDConfig utility again, and re-test the JTAG emulator. See section 2.15, Installing JTAG Emulator Software, above.**

2.1.7 Making Sure Processors are Not in Reset

All DSP processors must be removed from Reset before CCS can run! This is imperative – JTAG emulation will fail if the processor any DSP Reset line is active; CCS will abort and produce a general hardware failure error message in that case.

If the SigC5/6xxx hardware was booted in stand-alone mode, then CCS can now be run at any time.

If the SigC5/6xxx hardware was not booted in stand-alone mode, then you can run any Signalogic DirectDSP® or Hypersignal® program to initialize the DSP and prepare it for JTAG use. See section 5 below, Building Host Application Software, for Visual Studio, MATLAB, and Hypersignal software examples.

2.1.8 Running CCS

To run CCS, click on CCS icon on the desktop or go through the Start menu.

There should be no errors at start-up. If a dialog box appears with the error message “Cannot initialize target DSP,” then verify that the driver selected is the correct one for the processor type, and that the Port I/O field for the configuration matches that of SDConfig.

Once CCS is running, all JTAG debugging functionality will be enabled for the board entered in step 1. For more advanced SCI development options and common CCs debugging functions, see section 4 below, Building DSP Application Software, and section 6, Developing and Debugging DSP Code.

2.2 Installing Signalogic Software

Follow printed install instruction sheet included with the Signalogic software CD, then proceed to specific sections below.

Also, the following links may be helpful depending on specific type of software or hardware.

http://www.signalogic.com/index.pl?page=tech_support#Getting_Started

ftp://ftp.signalogic.com/documentation/Boards/SigC67xx/SigC67xxSetup.PDF

ftp://ftp.signalogic.com/documentation/Boards/SigC67xx/VDS_Users_Guide.pdf

ftp://ftp.signalogic.com/documentation/DirectDSP/Users 20Guide/

ftp://ftp.signalogic.com/documentation/DirectDSP/Reference 20Guide/

ftp://ftp.signalogic.com/documentation/SCI/C5xC6xCCS20JTAG.PDF

2.2.1 Electronic Format Software Upgrade

If you are receiving a zip file by e-mail for C5xxx or C6xxx SCI software, the following procedures are typical.

1) Before attempting to rebuild, unzip C5xxx files to the \dspcode\c5xxx subdir, create a Standard subdir under that, and move the Standard.* files to there. Perform the same step for C6xxx files.

3) It may be that some of the files in the .zip are “shared” files between C5xxx and C6xxx, in which case they should be installed at the \dspcode subdir level. If that is the case, the text accompanying the .zip file will contain instructions so indicating.

4) Files in the .zip file are read-only (due to version control software); you may need to reset file attributes before using or add to version control software as needed.

5) Hypersignal-Macro, DirectDSP, or other related driver update files may be included in the .zip. If so, then unzip additional files as follows:

 Copyright © Signalogic 2003-2005

Files	Unzip Target Subdir
hsmacro.exe hsmacro.ovr tmsc54xx.out tmsc5xx.out tmsc6x11.out	\hsmacro
hwlib*.dll enmgr*.dll hwmgr*.dll pipe.vxd	\windows\system
hwc5x_pc104.sys hwc6x_pc104.sys hwc5x_pci.sys hwc6x_pci.sys iim6x.sys	\windows\system32

2.2.2 Electronic Format Software Upgrade for MELPe

MELPe software upgrades assume the target system still has a \dspcode\c5xxx\MELPe subdir from previous code updates. Please check the zip file carefully -- files may have to updated at all levels: on the \dspcode subfolder, c5xxx subfolder, or \dspcode\c5xxx\MELPe subfolder. The zip file will contain path information for each file.

After unzipping, follow the steps below in section 4.1 below, CCS Project Build & Run Sequence.

2.2.3 Directory Structures

Below are shown directory structures for both DSP software and Host software after successful installation of Signalogic software CDs.

DSP SCI Software Directory Structure

After installing the Signalogic software CD, you should have the following subfolder organization for SCI software (Source Code Interface software), either for C5xxx or C6xxx DSP processors:

```
root dir
     |
     |____dspcode
            |
            |______C5xxx
            |
            |______C6xxx
                     |
                     |___SCI        <--- SCI build; higher performance,
                     |                   smaller memory footprint;
                     |                   DSP/BIOS not included
                     |
                     |___DSP_BIOS   <--- DSP/BIOS build; combines SCI code
                     |                   with Texas Inst RTOS; suitable for
                     |                   for larger projects needing tasks
                     .                   and intertask communication
                     .
                     . other / application specific projects
                     .
```

Although the above example is “expanded” to show C6xxx SCI structure, the structure is identical for C5xxx.

The above structure is required to successfully to build CCS projects with Signalogic-provided SCI software. It also matches the in-house Visual Source Safe version control format used at Signalogic, which makes it easier for Signalogic engineers to provide customer technical support updates. We recommend that you adhere to the above folder layout in all cases, as it will make it a lot easier for us to support you (thanks).

 Copyright © Signalogic 2003-2005

Win9x and WinXP Host Software Directory Structure

In addition to the SCI software directory structure shown above, there should be the following default directories, as listed below:

```
root dir
     |
     |_____Hsmacro
     |         |
     |         |
     |         |______Waveform
     |
     |
     |_____DirectDSP
               |
               |______Hwlib
               |
               |______RTC
                       .
                       .
                       .
```

 Copyright © Signalogic 2003-2005

Linux Host Software Directory Structure

In addition to the SCI software directory structure shown above, there should be three directories under the main extraction directory, as listed below:

1. SigFModule
2. EngMgr
3. HwMgr

By default, the main extraction directory is:

```
/usr/local/share
```

Note that the SigFModule folder contains files for both Hardware Library shared object module and the LKM driver.

2.2.4 WinXP Driver Install Instructions

Below are WinXP driver install instructions. Currently the drivers are not on the standard software install CD, so the manual installation procedure is provided.

Some WinXP drivers provided by Signalogic are Plug-N-Play and some require manual installation, depending on the board type and production status of the board. They have been tested under Win2K and WinXP; these drivers do not use the few Windows API functions that are not fully supported by both Win2K and WinXP.

2.2.4.1.1 Installing PC/104 and PTMC Board WinXP Drivers

Currently, WinXP drivers for PC/104 and PTMC boards must be installed in “manual mode”; i.e. these drivers are not Plug-N-Play.

The instructions below assume that when you first boot your PC with a PC104 or PTMC board installed, you will not install a driver from CD at that time. This will cause the warning indicated in step 4 below to appear in the Device Manager display

For PTMC boards using a PMC-to-PCI adapter board, when the PC boots up, it should indicate that new PCI hardware (the PCI board) has been found. You should see a dialog box that says "PCI Co-processor CPU" and prompts to search for a driver for the board. In response to that dialog, press:

```
Next (New Hardware Found)
Next (to Search for Drivers)
Next (Search on Floppy Disk or Other Location Ok)
Next (Unable to Locate Driver)
Finish
```

To complete driver installation in manual mode, below are the steps to take after your PC has finished booting.

1. Right-click on My Computer Icon on the desktop and select Properties menu item.

2. Click on “Hardware” tab.

3. Click the “Device Manager” button

4. You should see an item labeled “PCI Co-processor CPU” with a large yellow exclamation mark warning to it, that should indicate your M6x or SigC5/6xxx PTMC board. The warning occurs when new hardware is found at boot-time, but a driver is not installed at that time (see initial comments above).

5. Right-click and select “Properties” menu item.

6. Go to Driver tab and click the “Update Driver” button.

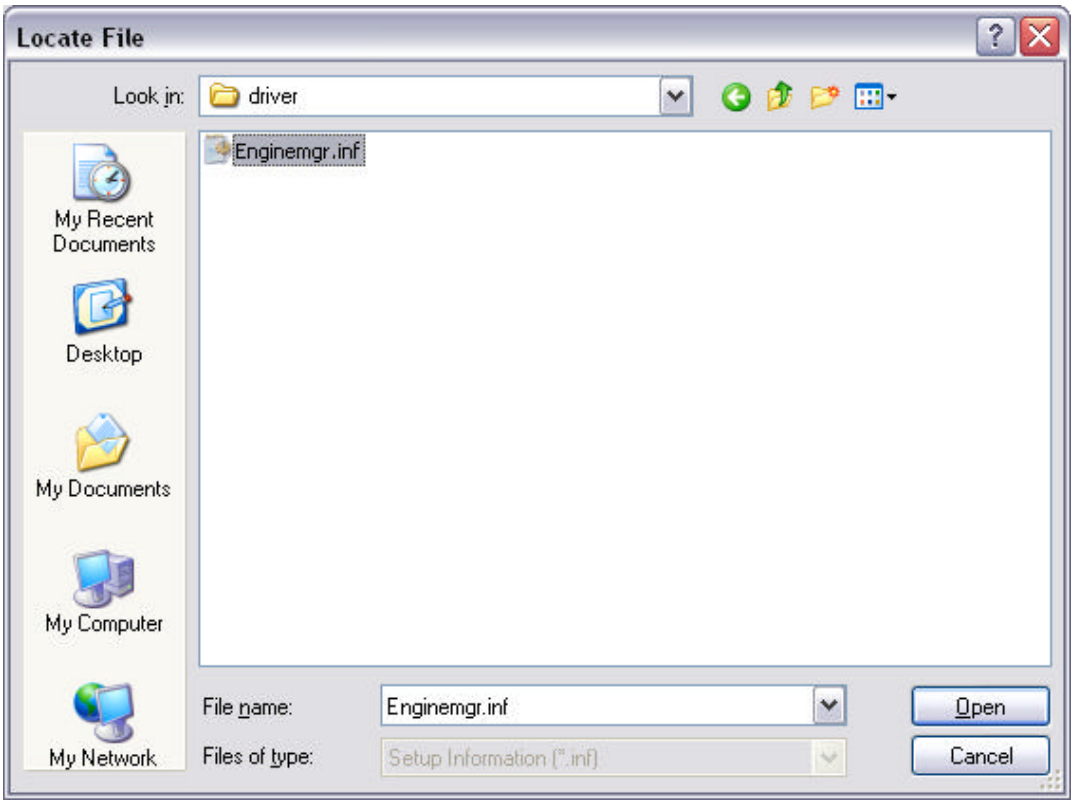
7. Click the “Next” button.

8. Choose “Display a list of known drivers for this device so that I can choose a specific driver” and click the “Next” button. You should see the following display:

9. Choose “Other Devices” and click next. You should see the following display.

Add Hardware Wizard
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Manufacturer
(Standard Infrared Port)
(Standard Modem Types)
(Standard port types)
(Standard system devices)
Model
Serial Cable using IrDA Protocol
This driver is digitally signed.
Tell me why driver signing is important
Have Disk...
< Back
Next >
Cancel

10. Click the “Have Disk” button. You should see the following display:



11. Select your subdir that contains Signalogic driver files (.inf and .sys files). This will be either a folder on the CD supplied by Signalogic, or a folder on your PC where you have downloaded driver files obtained from Signalogic. In the latter case, the folder should normally be:

```
\winnt\system32
```

12. Select the appropriate .inf file, as shown below:

Driver File	Board Type
hwc5x_pc104.inf	SigC5xxx PC/104
hwc6x_pc104.inf	SigC6xxx PC/104
hwc5x_pci.inf	SigC5xxx PTMC
hwc6x_pci.inf	SigC6xxx PTMC
iim6x.inf	M6x PCI

Note that .sys files with the same name should be located in the same folder as each .inf file. For example, if you have the hwc6x_pc104.sys file in the same location as the .inf file, WinXP will find and install it.

Click the “Finish” button on the next display to complete installation of the board driver.

13. Reboot your system and run Signalogic demo and example programs as needed.

 Copyright © Signalogic 2003-2005

DSP Code Location

For SigC6711-PC104 boards, make sure the following files are located correctly:

```
C:\hsmacro\tmsc6x11.out
C:\hsmacro\dskc6xhi.out
```

For SigC6713-PC104 boards, make sure the following files are located correctly:

```
C:\hsmacro\tmsc6x13.out
C:\hsmacro\dskc6xhi.out
```

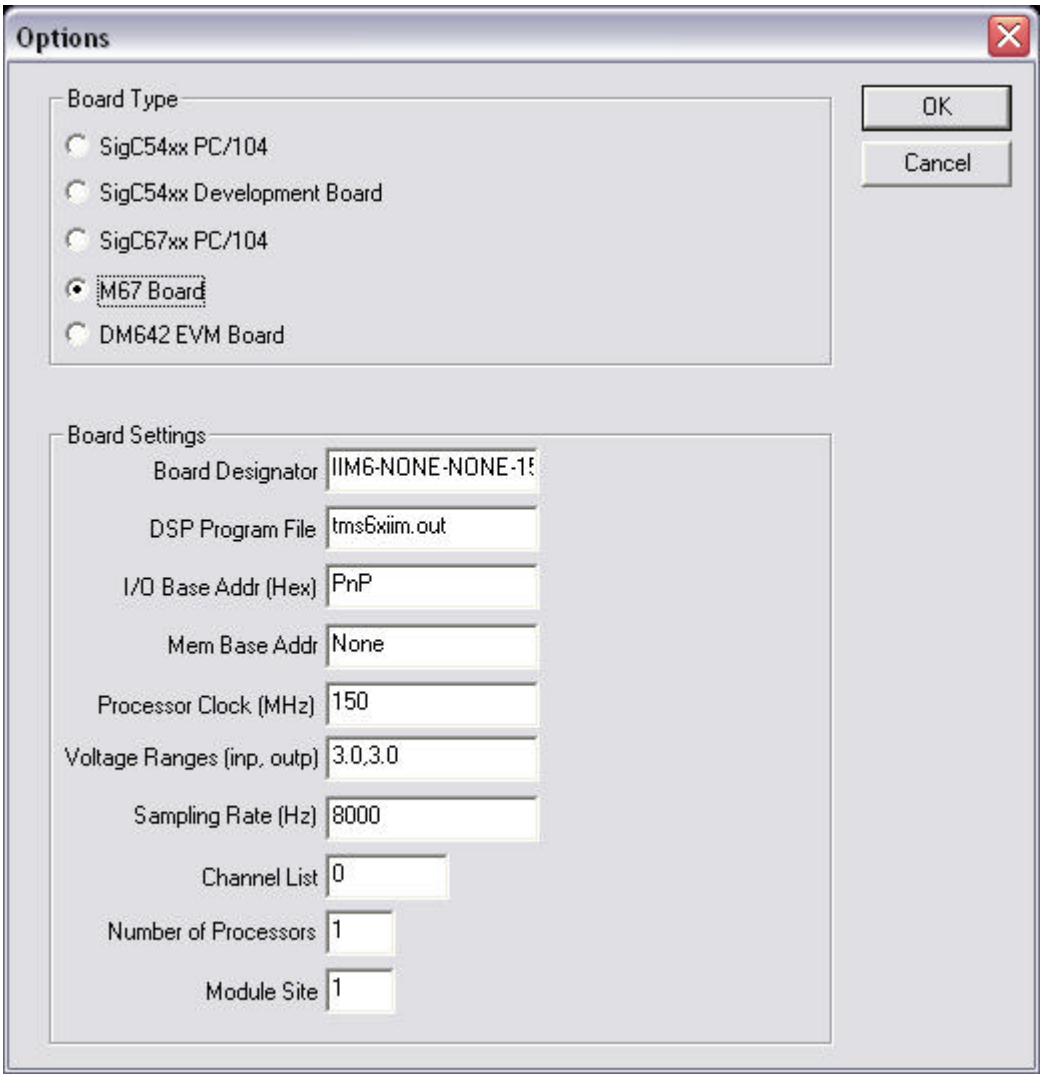
For SigC64xx-PTMC boards, make sure the following files are located correctly:

```
C:\hsmacro\tmsc64xx.out
C:\hsmacro\c64xxhi.out
```

In the above file lists, the first file is the default SCI executable and the second is the "talker", or host interface driver. For more information, see sections 4.1.1, SCI Builds, and 4.1.4, Talker – Host Interface Driver.

Test Programs

To run the digital scope test/demo program (for PC104 boards) or PTMC board test program…



Choose SigC6415-PTMC Board

2.2.4.1.2 Installing M6x Board WinXP Drivers

The M6x (M67 and M62) board driver is Plug-N-Play. The first time you plug in the board, WinXP will detect it and ask for the appropriate driver. You may use the Signalogic-provided files:

```
iim6x.inf
iim6x.sys
```

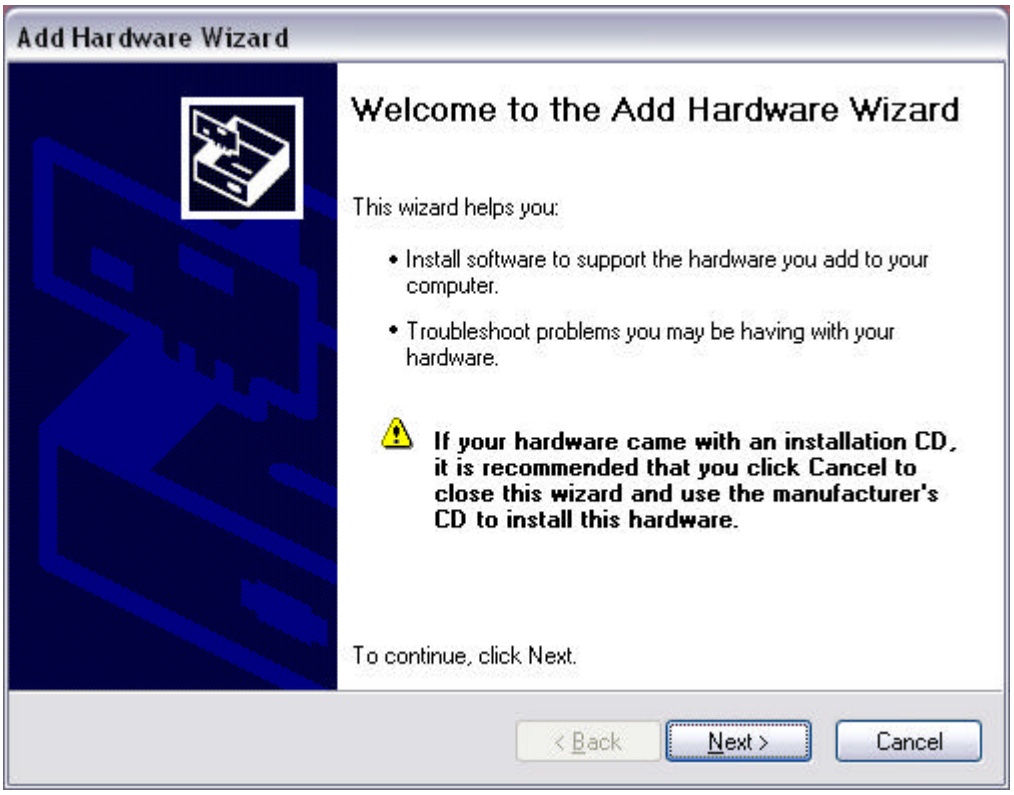
After installing the driver, you must reboot the computer to allow the driver to work correctly.

 Copyright © Signalogic 2003-2005

Installing Engine Manager for M6x

Engine Manager is a driver for receiving status and error messages from the driver, since there is no real hardware here, the system won't detect it, it has to be installed manually. This step should be done after installing the M6x board driver as explained in the previous section.

Start → Control Panel → Add Hardware


Add Hardware Wizard
Welcome to the Add Hardware Wizard
This wizard helps you:
• Install software to support the hardware you add to your computer.
• Troubleshoot problems you may be having with your hardware.
If your hardware came with an installation CD, it is recommended that you click Cancel to close this wizard and use the manufacturer's CD to install this hardware.
To continue, click Next.
< Back
Next >
Cancel

Click the "Next" button.

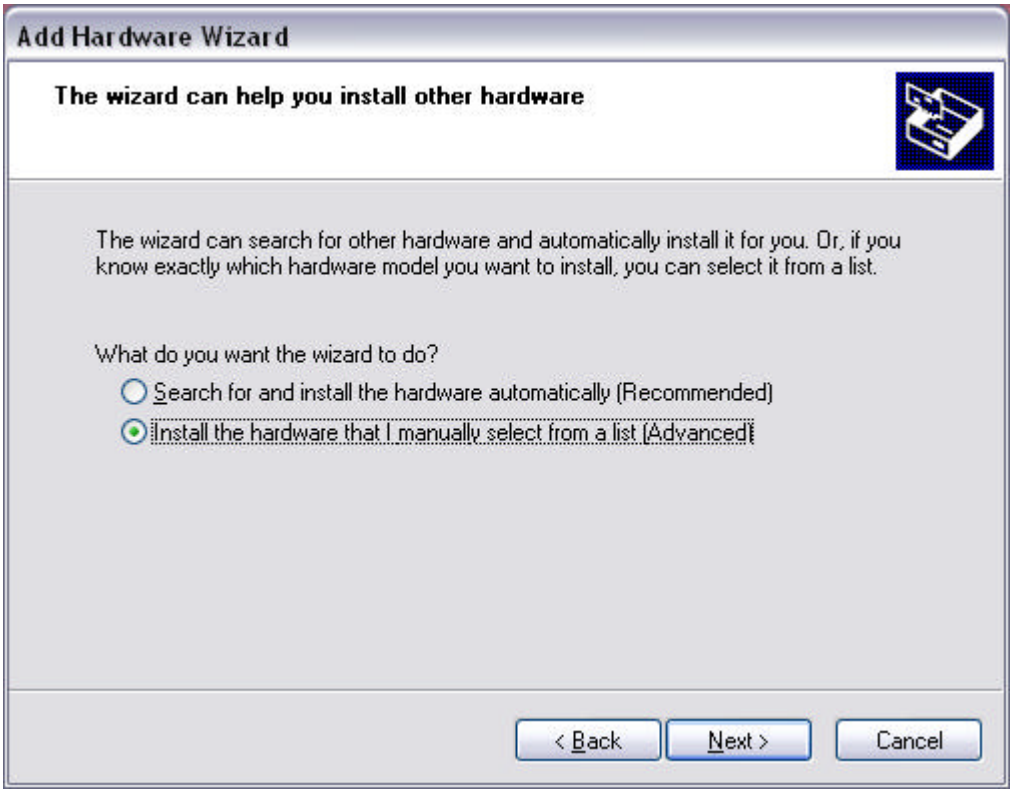
 Copyright © Signalogic 2003-2005

Add Hardware Wizard
Is the hardware connected?
Have you already connected this hardware to your computer?
Yes, I have already connected the hardware
No, I have not added the hardware yet
< Back
Next >
Cancel

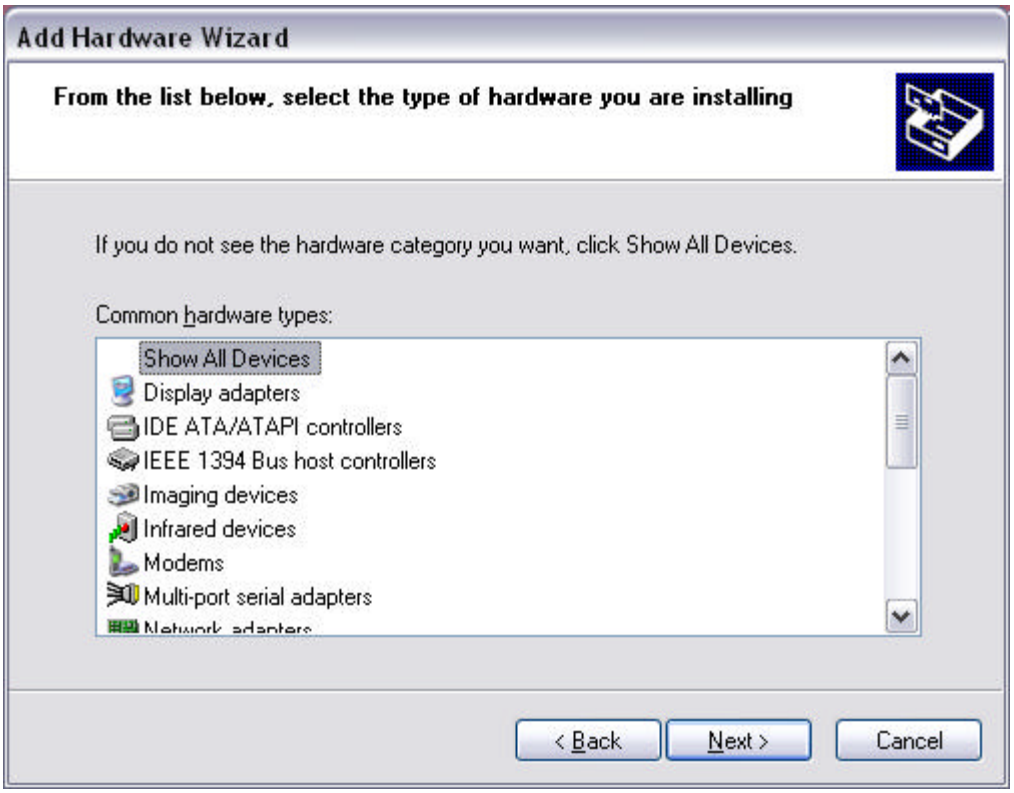
Select “Yes, I have already connected the hardware” and click the “Next” button.

Add Hardware Wizard
Is the hardware connected?
Have you already connected this hardware to your computer?
Yes, I have already connected the hardware
No, I have not added the hardware yet
< Back
Next >
Cancel

Select “Add a new Hardware device”, and click the “Next” button.


Add Hardware Wizard
The wizard can help you install other hardware
The wizard can search for other hardware and automatically install it for you. Or, if you know exactly which hardware model you want to install, you can select it from a list.
What do you want the wizard to do?
Search for and install the hardware automatically (Recommended)
Install the hardware that I manually select from a list (Advanced)
< Back
Next >
Cancel

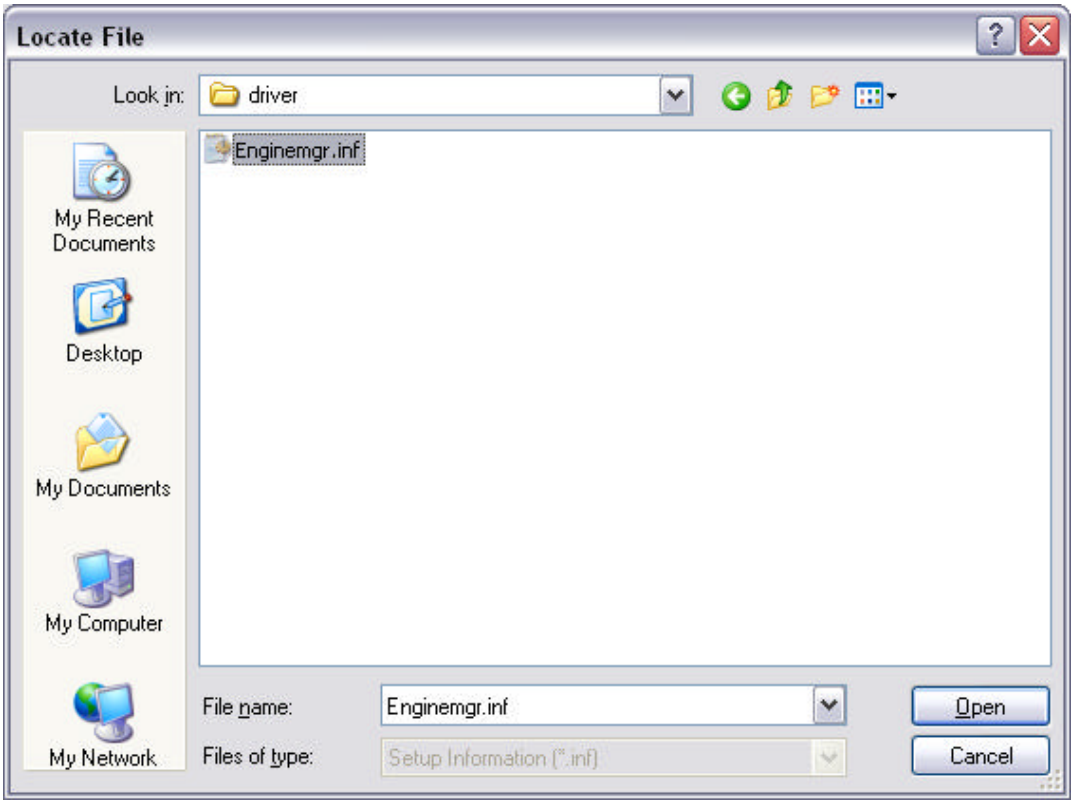
Choose “Install the hardware that I manually select from a list (Advanced)”, and click the “Next” button.


Add Hardware Wizard
From the list below, select the type of hardware you are installing
If you do not see the hardware category you want, click Show All Devices.
Common hardware types:
Show All Devices
Display adapters
IDE ATA/ATAPI controllers
IEEE 1394 Bus host controllers
Imaging devices
Infrared devices
Modems
Multi-port serial adapters
< Back
Next >
Cancel

Choose “Show All Devices” and click the “Next” button.

Add Hardware Wizard
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Manufacturer
(Standard Infrared Port)
(Standard Modem Types)
(Standard port types)
(Standard system devices)
Model
Serial Cable using IrDA Protocol
This driver is digitally signed.
Tell me why driver signing is important
Have Disk...
< Back
Next >
Cancel

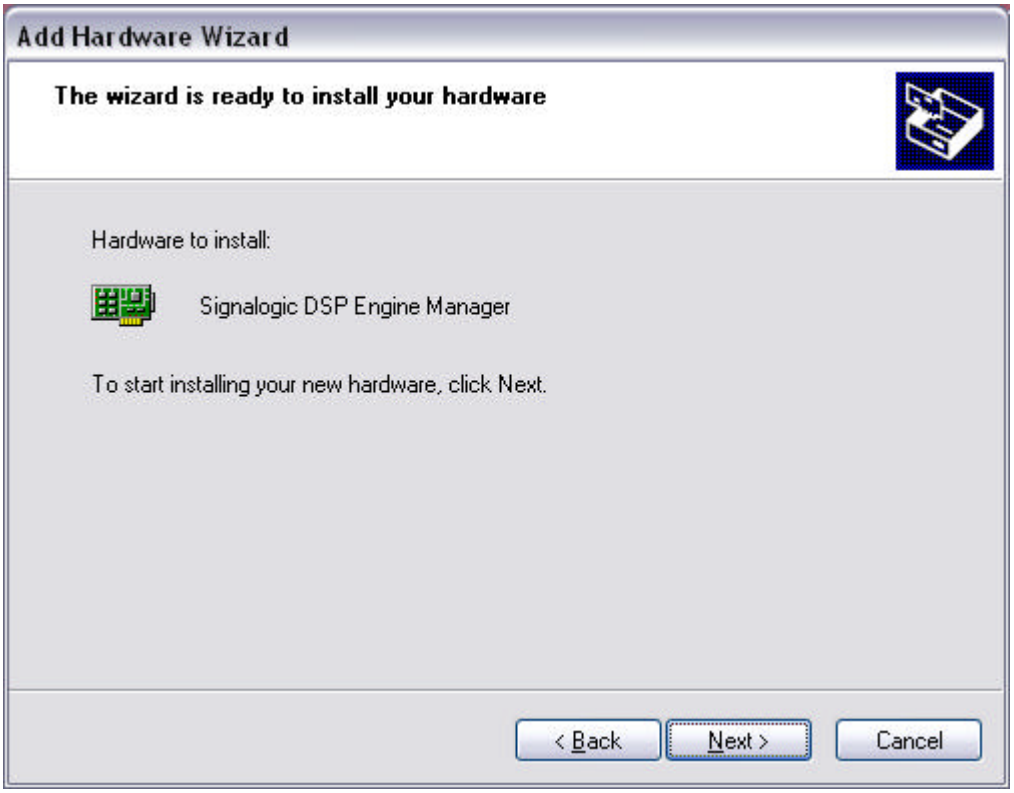
Click the “Have Disk” button.



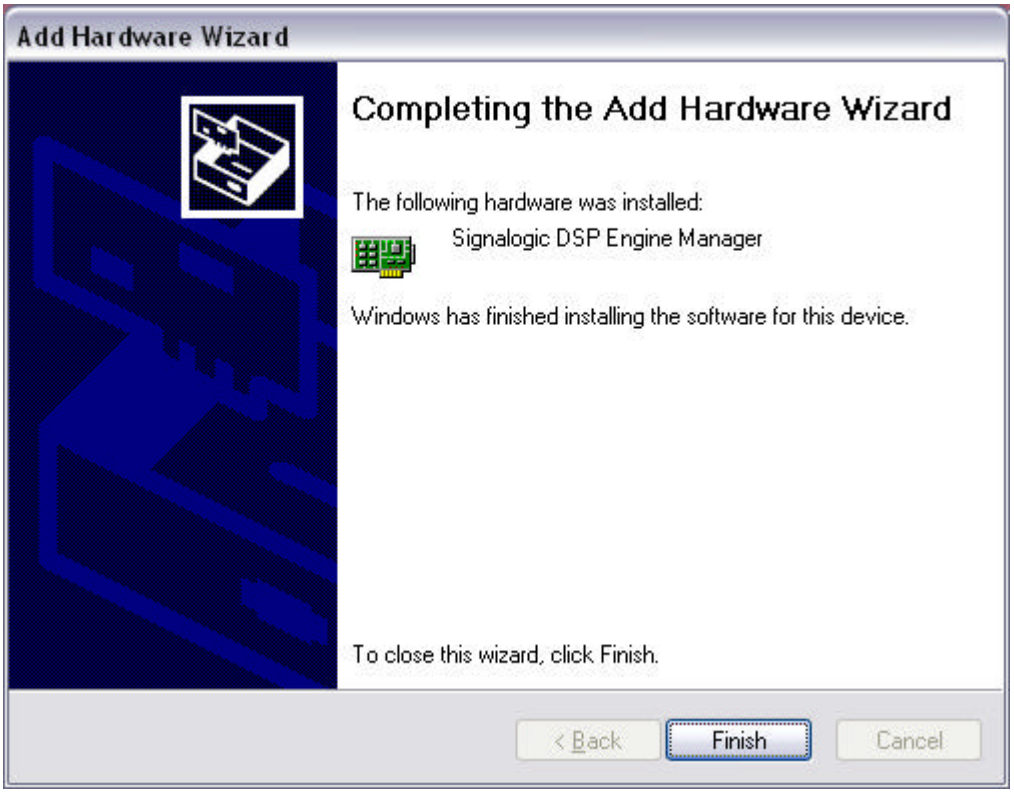
Browse for Enginemgr.inf and click the “Open” button, then click the “Next” button on the previous display.

Add Hardware Wizard
Select the device driver you want to install for this hardware.
Select the manufacturer and model of your hardware device and then click Next. If you have a disk that contains the driver you want to install, click Have Disk.
Model
Signalogic DSP Engine Manager
This driver is not digitally signed!
Tell me why driver signing is important
Have Disk...
< Back
Next >
Cancel

Click the “Next” button.


Add Hardware Wizard
The wizard is ready to install your hardware
Hardware to install:
Signalogic DSP Engine Manager
To start installing your new hardware, click Next.
< Back
Next >
Cancel

Click the “Next” button.


Add Hardware Wizard
Completing the Add Hardware Wizard
The following hardware was installed:
Signalogic DSP Engine Manager
Windows has finished installing the software for this device.
To close this wizard, click Finish.
< Back
Finish
Cancel

If you have the enginemgr.sys file in the same location as the inf file, WinXP will find and install it. Click the “Finish” button to complete installation of Engine Manager.

DSP Code Location

Make sure the following files are located correctly:

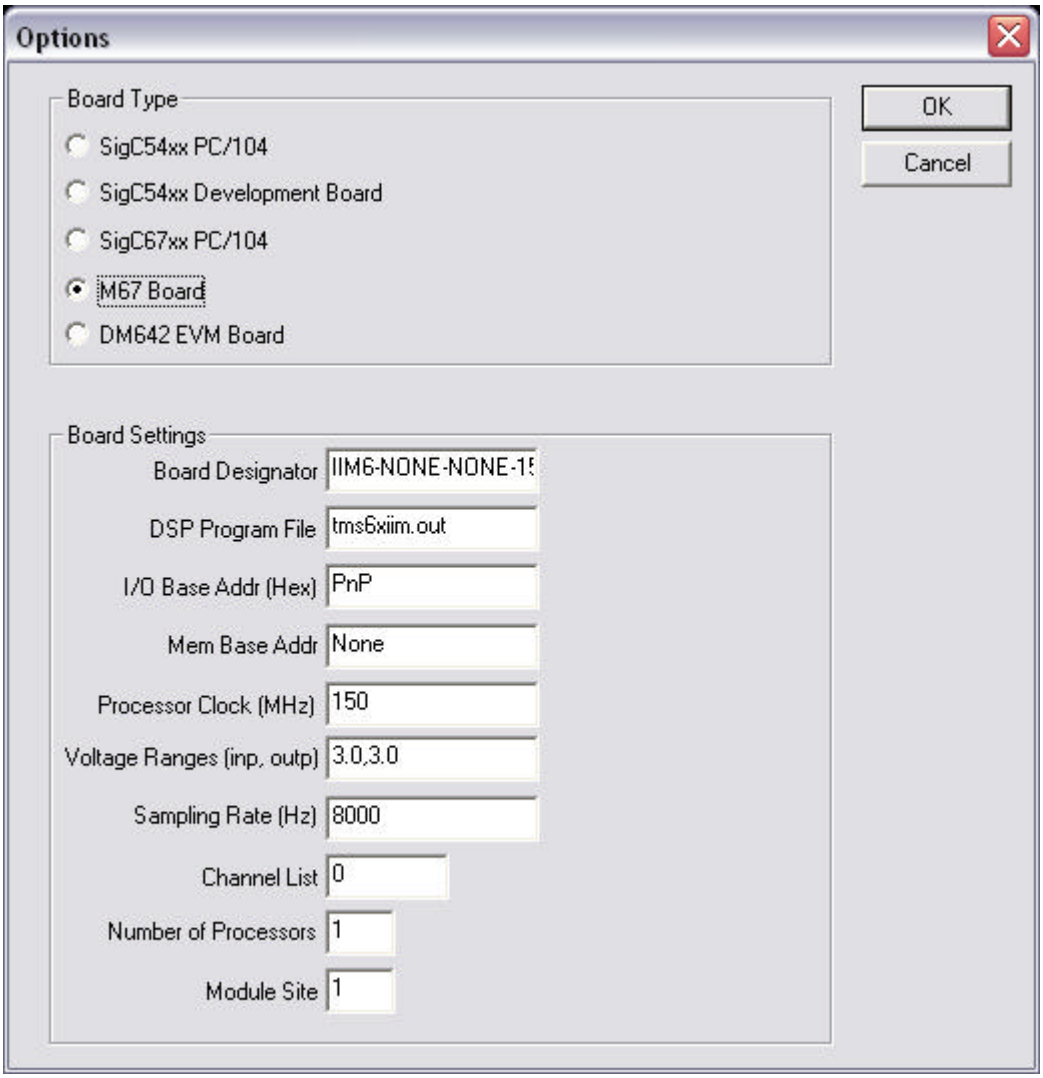
```
C:\hsmacro\c6201hi.out
C:\hsmacro\tms6xiim.out
```

The first file above is the default SCI executable and the second is the “talker”, or host interface driver. For more information, see sections 4.1.1, SCI Builds, and 4.1.4, Talker – Host Interface Driver.

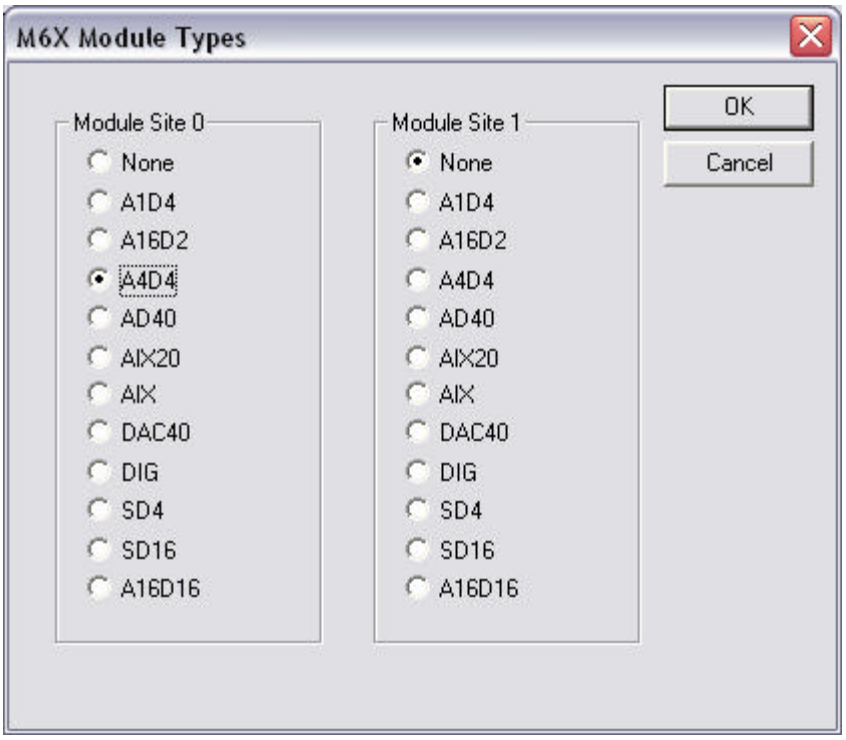
Digital Scope Test Program

To run the digital scope test/demo program…

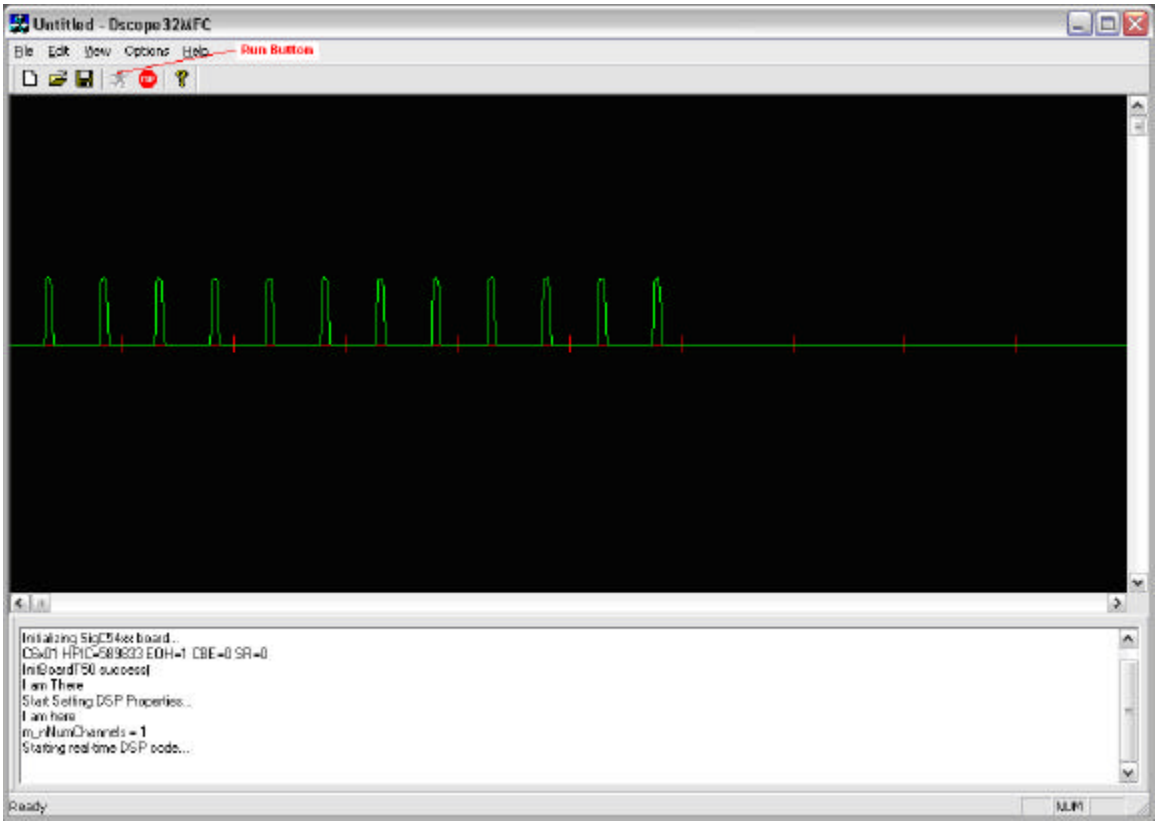
 Copyright © Signalogic 2003-2005



Choose M67 Board



Choose A4D4 for module site 0 and None for Module Site 1. Click Ok and Ok.



 Copyright © Signalogic 2003-2005

Click on the Run Button to run the board. You should see the following messages in the status display and if you are feeding something into the analog input of channel 0, you should see the display of the input, otherwise a flat line.

Click on the stop button to stop running the board.

Installing WinXP Drivers from CD

2.2.5 Linux Driver Install Instructions

Following are instructions for installing Linux drivers for supported board types.

2.2.5.1 SigC5xxx and SigC6xxx PC104 Linux Driver Install Instructions

To install the SigC5xxx and SigC6xxx PC104 board loadable kernel module (LKM) driver, follow the steps below:

1) Log-in as root.

2) Install the loadable kernel module:

```
shell/> cd /usr/local/share/SigFModule/bin
shell/> insmod SigFM.o
```

3) Change back to the main extraction directory by entering:

```
shell/> cd ../..
```

To rebuild the loadable kernel module driver, follow the instructions in section 5.3.3 below, Rebuilding LKM Drivers.

NOTE: *currently the LKM driver must be installed each time the system is rebooted. However, in the future Signalogic engineers plan to make the driver load automatically, by editing the /etc/modules.conf file and adding "module probing" of the SigC5xxx or SigC6xxx board at boot time.*

2.2.5.2 SigC641x and SigC5561 PTMC Linux Driver Install Instructions

Following are preliminary install procedures for the SigC6415-PTMC card Linux driver.

1) Log-in as root.

2) Load the module either directly or with the following command:

```
insmod sigc641x.o
```

or with the makefile provided with the driver, as shown in Figure 2-12 below. If you use the insmod command, note the filename extention ".o" must be included.

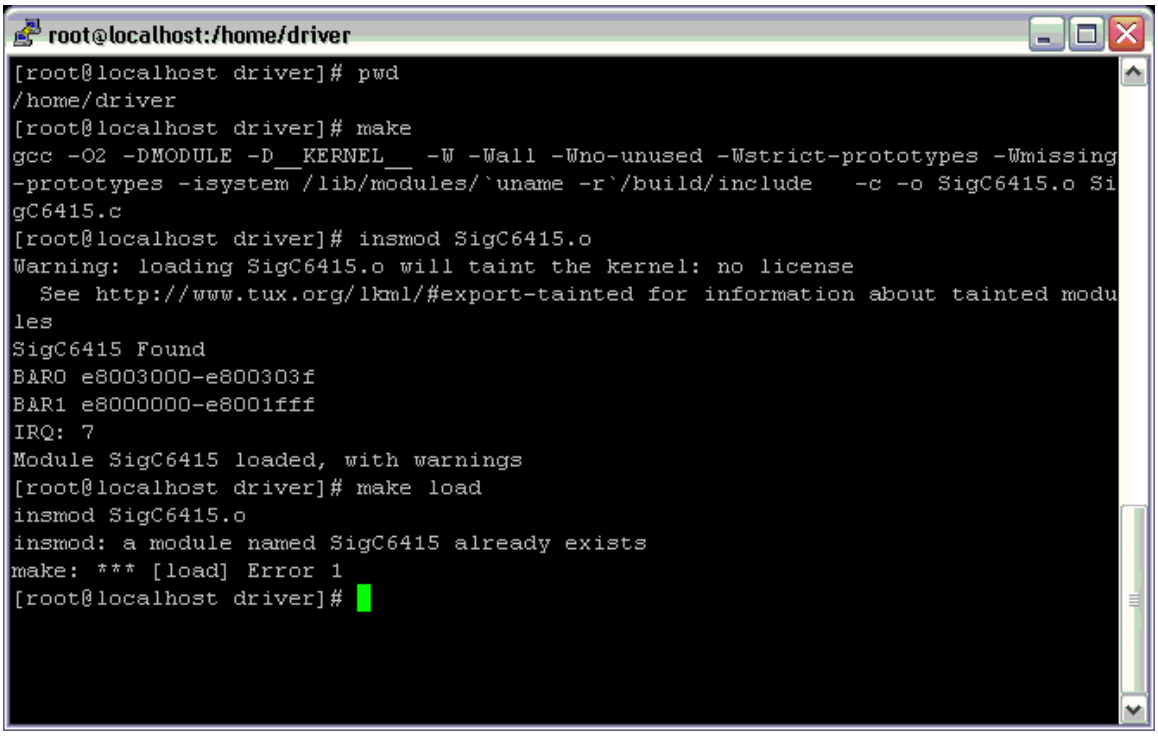


Figure 2-12, SigC641x and SigC5561 PTMC Card Driver Load

Note that the makefile will also load the driver if the parameter "load" is added:

```
make load
```

For more information on re-building PTMC card drivers, see section 5.3.3.1 below, Rebuilding the SigC641x and SigC5561 PTMC LKM Drivers.

The kernel tainting warning produced by the insmod command can be safely ignored; it signifies that the module does not have an open source license.

2) You can now run user applications that make use of the SigC641x PTMC device driver.

3) To remove the module, type the following command:

```
rmmod sigc641x
```

2.2.6 Hypersignal-Macro Software Board Support

Installing Hypersignal software from the standard Signalogic software CD.

 Copyright © Signalogic 2003-2005

Hypersignal supports a number of board types under Win9x and WinXP operating systems, but has no Linux support. Below is a table by board type.

Board Type	**Win9x**	**WinXP**
SigC5xxx-PC104	Y	Y
SigC6xxx-PC104	Y	Y
SigC64xx-PTMC	Y	N
M6x	Y	N
DSK 5402	Y	Y
DSK C6711	Y	Y

 Copyright © Signalogic 2003-2005

3 Hardware Installation

Please follow the instructions in sections 3.x below, depending on your specific type of hardware.

Also, the following links may be helpful depending on specific type of software or hardware that you have.

http://www.signalogic.com/index.pl?page=tech_support#Getting_Started

ftp://ftp.signalogic.com/documentation/Boards/SigC67xx/SigC67xxSetup.PDF

ftp://ftp.signalogic.com/documentation/Boards/SigC67xx/VDS_Users_Guide.pdf

ftp://ftp.signalogic.com/documentation/DirectDSP/Users 20Guide/

ftp://ftp.signalogic.com/documentation/DirectDSP/Reference 20Guide/

ftp://ftp.signalogic.com/documentation/SCI/C5xC6xCCS20JTAG.PDF

3.1 Installing SigC5xxx and SigC6xxx PC/104 Hardware

SigC5xxx and SigC6xxx PC/104 boards can be installed either inside a PC or in a stand-alone stack.

3.1.1 Installing PC/104 Boards in a PC

When installing Rev C1 PC/104 boards in a PC, an ISA slot must be available. Rev D and later boards can be installed either in an ISA slot or a PCI slot. Figure 3-1 below shows a Rev C1 PC/104 board connected to a "passive carrier card" that allows it to be inserted into an ISA slot in a standard PC:

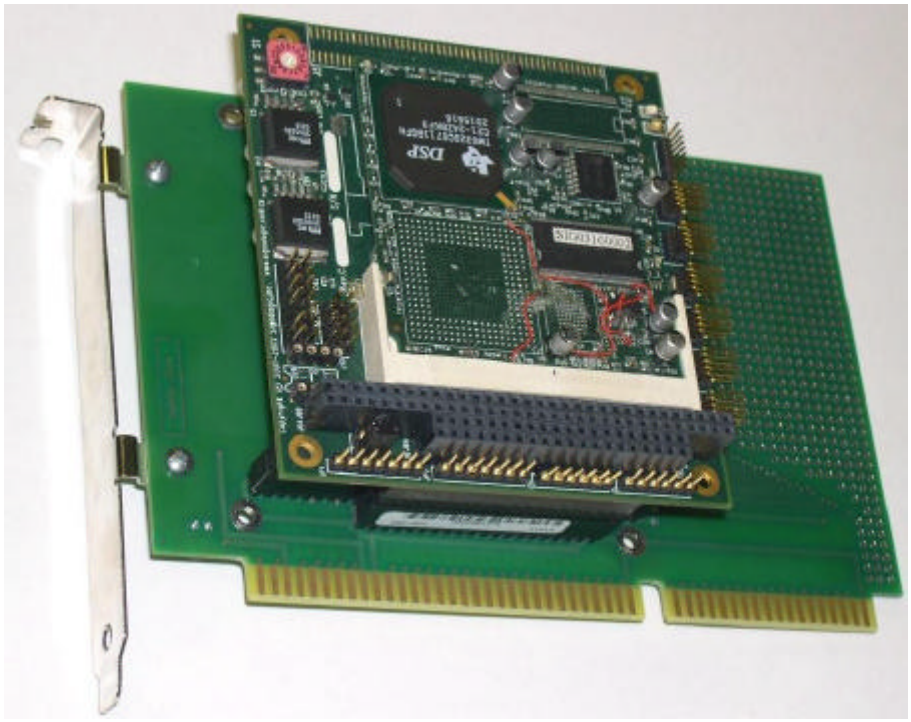


Figure 3-1, SigC6xxx-PC/104 Board Connected to Passive Carrier Card

3.1.2 Installing PC/104 Boards in a Stand-Alone Stack

Figure 3-2 below shows a SigC5xxx-PC/104 board installed in a stand-alone board stack

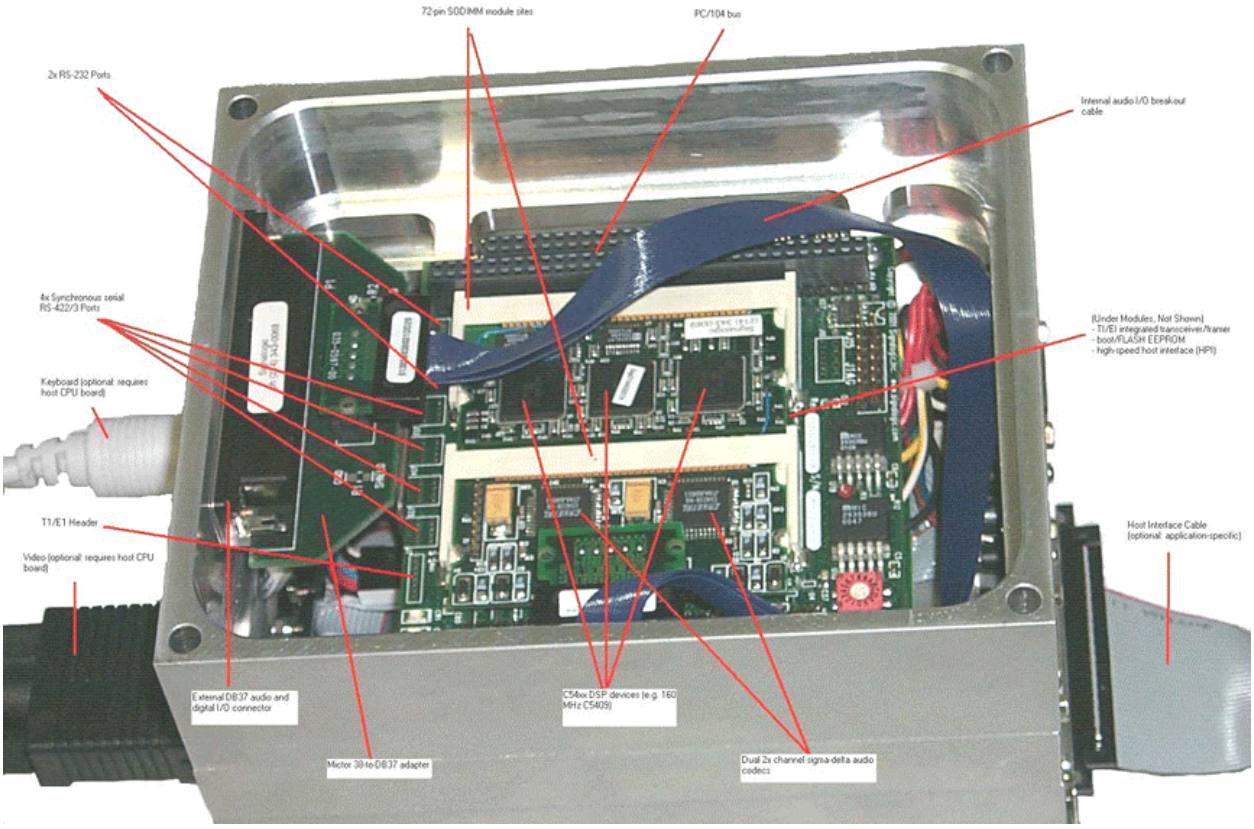

72-pin SODIMM module sites
PC/104 bus
2x RS-232 Ports
Internal audio I/O break-out cable
4x Synchronous serial RS-422/3 Ports
Keyboard (optional: requires host CPU board)
T1/E1 Header
Video (optional: requires host CPU board)
Host Interface Cable (optional: application-specific)
External DB37 audio and digital I/O connector
Mictor 38-to-DB37 adapter
C54xx DSP devices (e.g. 160 MHz C5409)
Dual 2x channel sigma-delta audio codecs

Figure 3-2, SigC5xxx-PC/104 Board Installed on a Stand-Alone Stack

In Figure 3-2 above, the SigC5xxx board is installed on top of the stack, and the stack is contained inside an enclosures suitable for embedded system use. In the pictures and diagrams below, the Signalogic hardware is assumed to be at the top of the PC/104 board stack for clarity explanation purposes, although that is not necessary in actual systems.

3.1.3 PC104 Board Installation Steps

Following are important hardware installation steps that should be performed and verified prior to applying power and running the system.

1. Base I/O Address Switch The SigC5xxx-PC/104 board I/O base address default value is 0x320. In some cases, this value may need to be changed to avoid conflict with another board in the system or stack. For example, in some PC or Pentium-compatible systems (such as Lippert Systems motherboards) the SigC5xxx-PC/104 board base address must be set to 0x340 to avoid conflict with other boards. Figure 3-3 below contains a picture that shows the location of the I/O base address switch (lower left of picture).

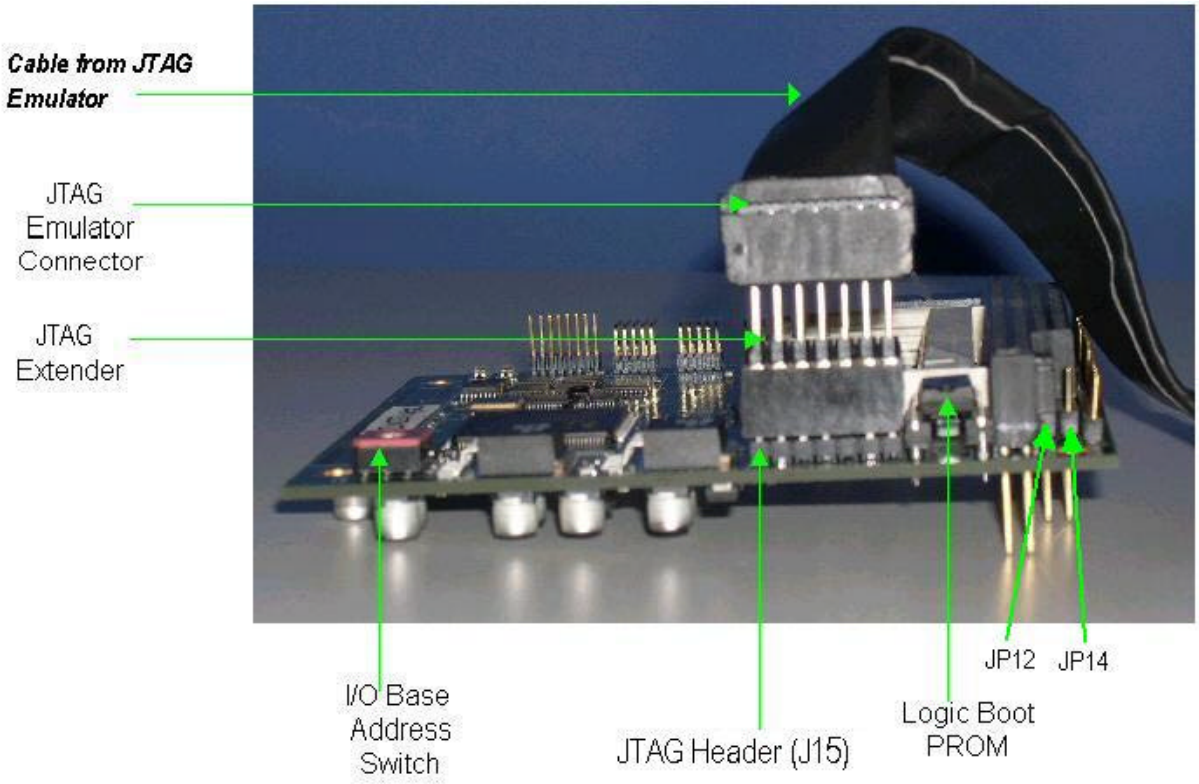


Figure 3-3, I/O Base Address Switch and JTAG Emulator Connection

To change or check the SigC5xxx board I/O base address, verify that the rotary hex switch on the board is set to the middle digit of the desired base I/O address. For example, for I/O base address 0x320 the switch should be set to “2”, and for I/O base address 0x340 the switch should be set to "4".

Another labeled board picture that might be helpful during installation is shown in Figure 3-4 below.

 Copyright © Signalogic 2003-2005

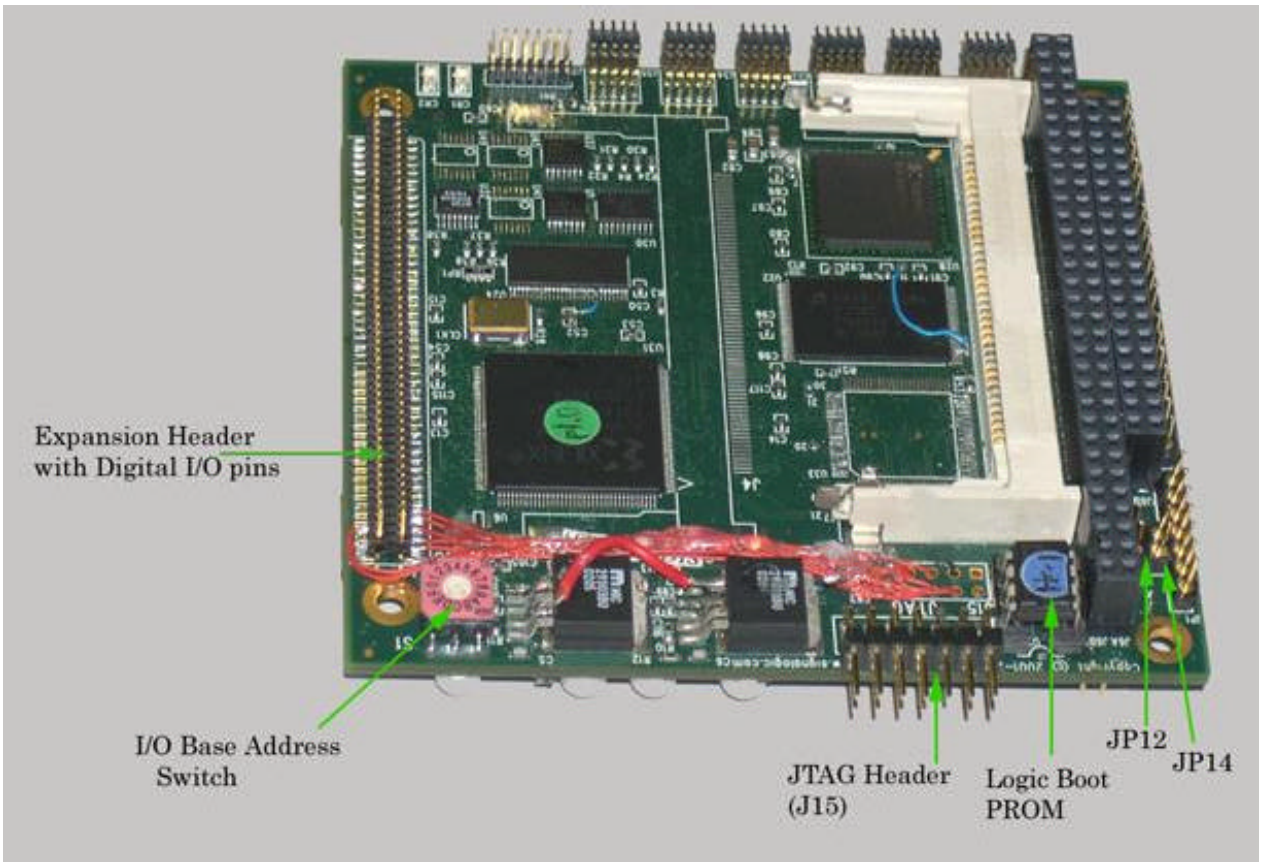


Figure 3-4, SigC6xxx-PC/104 Board with Expansion Header Installed

In the picture above, note the jumper wires installed to support digital I/O on a Rev C1 board -- Rev D and higher boards do not require those wire modifications. The board in the above picture is configured for a SigC67xx module (only one module site on top side), but the module is not installed in the picture. Note also that a right-angle JTAG header has been installed – you may want to order the board in that configuration for more convenient connection to a JTAG emulator – see additional comments below and also section 2.1.4 below, Installing XDS510PP JTAG Emulator.

2. **Jumpers** JP12 should be installed for logic boot from PROM. The only time JP12 is not installed is if a Xilinx "pod" is being used to load and boot the logic. However, this mode is not recommended and should be performed only by Signalogic engineering staff in a lab environment.

JP14 should be installed if stand-alone mode boot is required. Stand-alone mode boot causes onboard logic to perform the following functions:

a. Automatic module-type detection (see section 10.3, Logic Registers)

b. DSP processors are removed from Reset, but not run (idle mode)

c. PC104 board Flash EEPROM contents are checked for a valid "executable code" signature; if found, code is bootloaded into one or more DSPs and DSPs are placed into run mode

JP14 should not be installed if the board is being controlled by a host processor in the system, and/or DSP code is being downloaded by a host processor. However, in that case, before attempting to run CCS software, a test program, host program, or other host tool should be used to take the DSP processor(s) out of reset before invoking CCS.

For more information about EEPROM programming and run-time boot, see sections 10.7.2 and 10.7.3 below, EEPROM Programming Instructions and EEPROM Boot Flow.

3. Optional, if Using JTAG Emulator If you are using a JTAG emulator, then you should connect to J15 header on the PC/104 board, as shown in Figure 3-3 above. Note that you may need to use a small extender if the logic boot PROM is installed in a socket, otherwise the JTAG cable connector may interfere with the PROM or PROM socket. For production system boards, the logic boot PROM may be installed as a small surface mount device (the sm site is within the socket site; they occupy the same space on the board). Another alternative is to install a right-angle JTAG header, or request the board to be ordered from Signalogic that way, if it is acceptable to slightly exceed the mechanical form-factor and dimensions specified by the PC/104 specification. See the picture in Figure 3-4 above for an example.

Note also that the JTAG connector is keyed! Pin 6 is removed on J15 to match the keyed header provided by most JTAG emulators that support Texas Instruments DSP hardware. *Do not force the connector and try to connect the header in reverse!*

4. Optional, if Using SigSDn Audio Module If you are using a SigSDn audio module, note also the position of the audio I/O Mictor cable connector, connected directly to the SigSDn audio module (shown as installed in module site 4, or second module site on top side of the board, at bottom of picture in Figure 3-5 below).

 Copyright © Signalogic 2003-2005

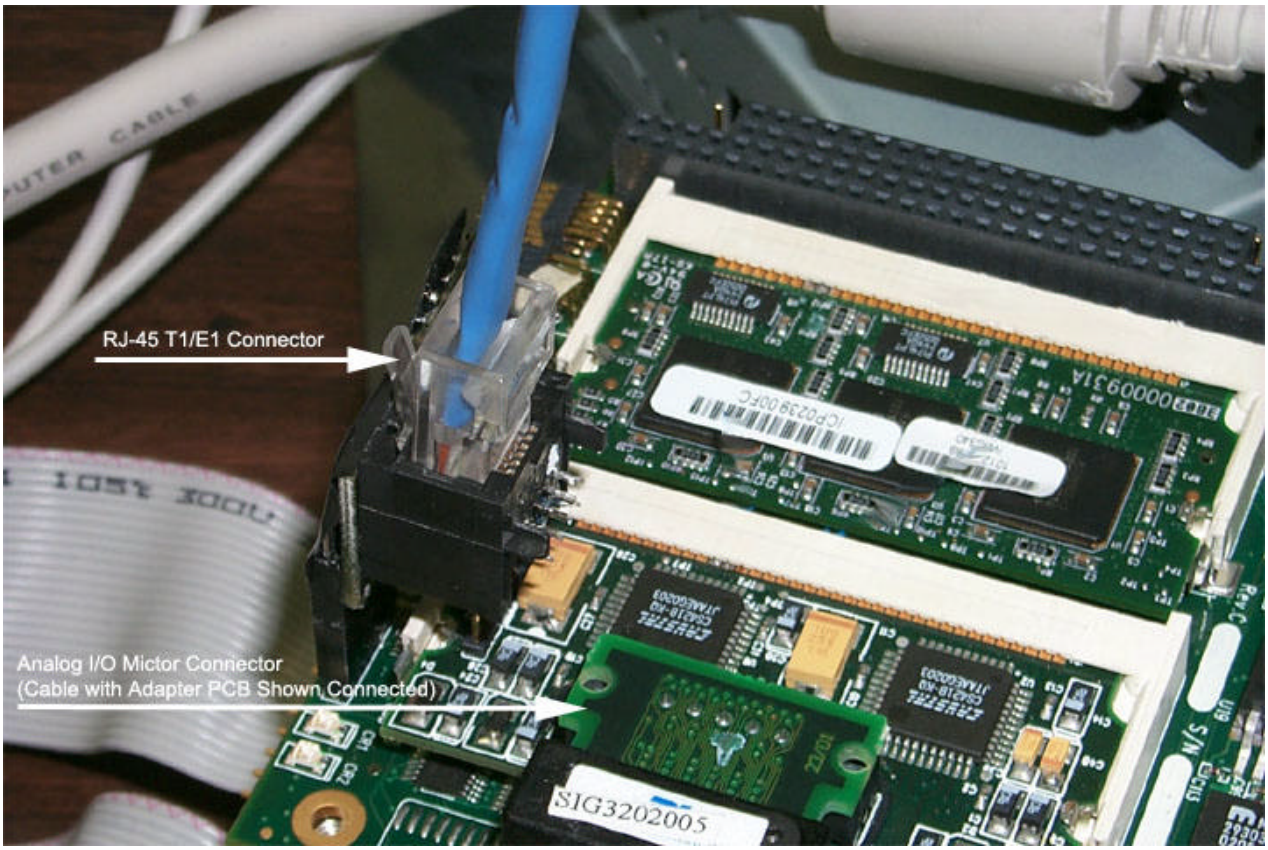


Figure 3-5, SigC5xxx-PC/104 T1/E1 and Audio Connections

5. <u>Optional, if Using T1/E1 Peripheral</u> In the above picture, note the position the RJ-45 / magnetics adapter PCB, mounted at left side of the SigC5xxx-PC/104 board. The E1 loopback cable and plug is shown inserted (blue cable). The plug cable is about 3" long with wires exposed at one end and has RXRING tied to TXTIP and RXTIP tied to RXRING.

For E1-audio loopback testing, be sure that the loopback plug is securely inserted into the RJ-45 adapter PCB. ***The loopback plug should be pushed all the way in*** and not allowed to rock back and forth.

6. <u>Optional, if Using DUART Peripheral</u> For DUART testing, ensure that the RS-232 adapter is securely inserted into the DB9 adapter PCB header at J35 (topmost RS-232 header on the right side of the board). J35 contains signal for DUART channel A; J34 contains signal for DUART channel B. See Figure 3-6 below for additional explanation.

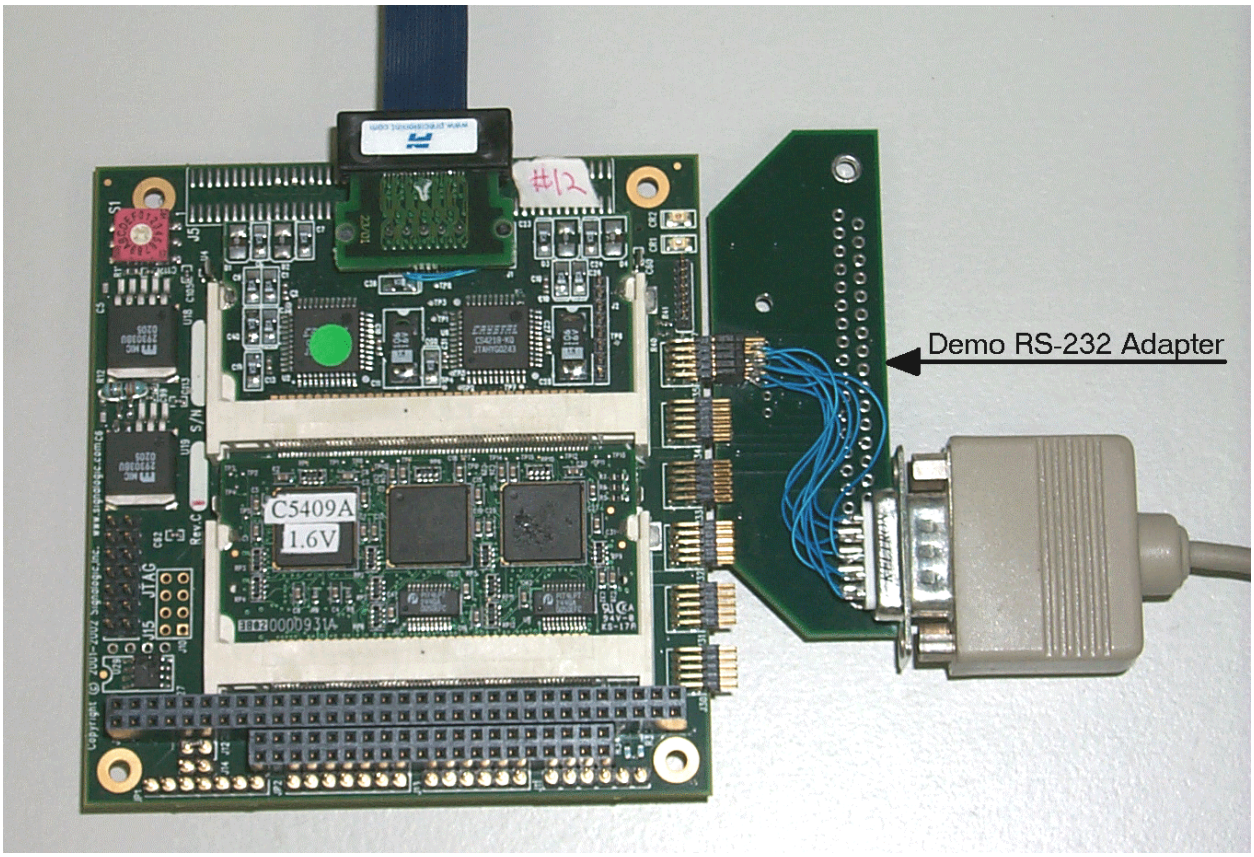


Figure 3-6, SigC5xxx-PC/104 with DUART adapter shown

For lab testing purposes, a DB9 female-to-female serial cable may be connected from the RS-232 adapter to a PC COM port. See Figure 3-7 below for additional explanation.

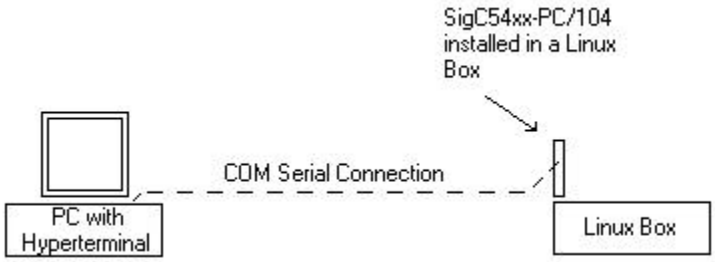


Figure 3-7, SigC5xxx-PC/104 with COM Serial Connection to Hyperterminal PC or other TTY Terminal Emulator

7. <u>Optional, if Using RS-422 Peripheral</u> RS-422 connectors are the first four (4) connectors shown along the right hand side of the board in the picture in Figure 3-6 above (J30-J33, before the UART port B connector, J34). In the picture below a “loopback connector” is attached to J30, or RS-422 port A. The loopback connector connects pins 2 and 3, or Rx and Tx data, of the connector. See section 10.9 below, Routing Logic, for more information..

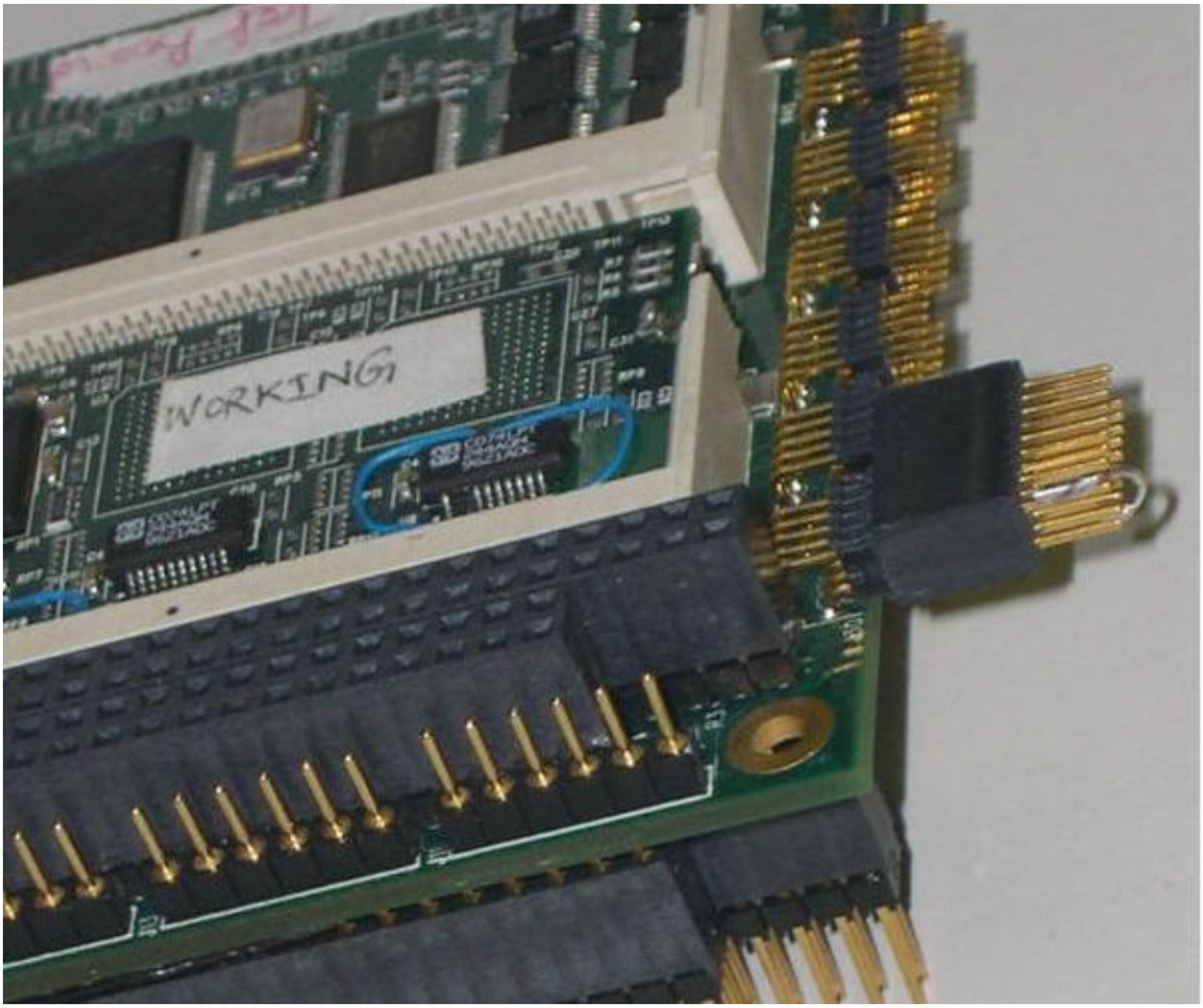


Figure 3-8, RS-422 Connectors with Loopback Installed on Port A

8. <u>Final Safety Checks</u> When the board is correctly configured, then perform the following crucial last-minute checks:

- *Ensure PC/104 pins are correctly inserted!! Look very closely. There should be no pins off by one row, or outside the connector*

- *Make sure the board is well-seated and pushed down firmly. The board should not be angled, or rock back and forth.*

It must be emphasized that if you plug the board in wrong it will likely be damaged due to a hideous and brutal condition, +12V going to a 3.3V IC pin, that will kill a logic device or other IC responsible for bus-interface on the board. In that case, necessary repairs are not covered under warranty!! So please be very careful during board installation – you have been warned.

9. Applying Power Immediately after applying power, you should see the board LED labeled CR1 illuminate, indicating normal and valid logic boot status. If CR1 is flashing then it indicates a logic or hardware error condition. If you do not see this LED illuminate in any way, then turn power off without delay and examine the board and carrier card insertion carefully.

3.1.4 Additional Notes For Stand-Alone PC/104 Board Systems

1. For some Pentium PC/104 main boards, the hard drive (HD) connector can be very unstable. For example, on a Lippert CoolRunner II board, the HD cable will come loose at the slightest twitch. ***Watch out for this!!*** The cable should be securely attached and pushed down fully onto the main board.

2. Note that the video on the Lippert Systems main board has been set to 800 x 600 x 4 due to the low amount of memory (less than 2 Mbyte) reserved for video display. This may make some GUI dialogs and screen displays look "nearly monochrome", but it's the best that can be done given system limitations.

3.1.5 Power Consumption Reduction PC104 Board Modifications

Below is a list of power consumption savings modifications that can be specified when ordering Rev C1 PC/104 boards -- with exceptions of item 5 (always possible) and item 6 (application-specific). For items 1-4, existing Rev C1 boards may also be upgraded.

For Rev D and higher boards, the modifications are incorporated into the board design. Rev D and higher boards also use switching regulators for 3.3V and module site Vcc Core, which improves power consumption efficiency significantly for boards configured with extensive I/O options and/or multiple processors.

1) Jumper input to module site Vcc Core regulator on the PC104 board (U18) to be 3.3V instead of 5V; i.e. use output from 3.3V regulator (U19). This modification is acceptable in 1 to 2-processor situations, but is not recommended for 3 or more processors, which would require greater output on the module site Vcc Core rail. Note that for some module types, actual on-module Vcc Core voltage (ranging from 1.2V to 1.8V depending on module type) is derived on-module from the edge connector value, typically 2.5V in those situations.
2) Separate regulator for RS-422 drivers is installed. Regulator is under control of enable/disable bit in logic register Power Control register.
3) Separate regulator for RS-232 driver is installed. Regulator is under control of enable/disable bit in logic register Power Control register.
4) A separate regulator for logic boot PROM is installed that allows power to be turned off and PROM is electrically isolated after logic boot. This occurs automatically under logic control.
5) DSP clock may be controlled by DSP software; for example, slow DSP clock when system is in an idle or other less processing intensive operating mode.
6) Program PC104 board Routing Logic PLD with minimal logic version customized per application. Eliminates all unused pin definitions and logic.

3.2 Installing SigC55xx and SigC64xx PTMC Hardware

SigC5xxx and SigC6xxx PTMC cards can be installed either inside a PC or a cPCI carrier board, as shown in sections 3.2.1 and 3.2.2 below.

Following is a picture of the SigC6415 board showing important connectors and headers:

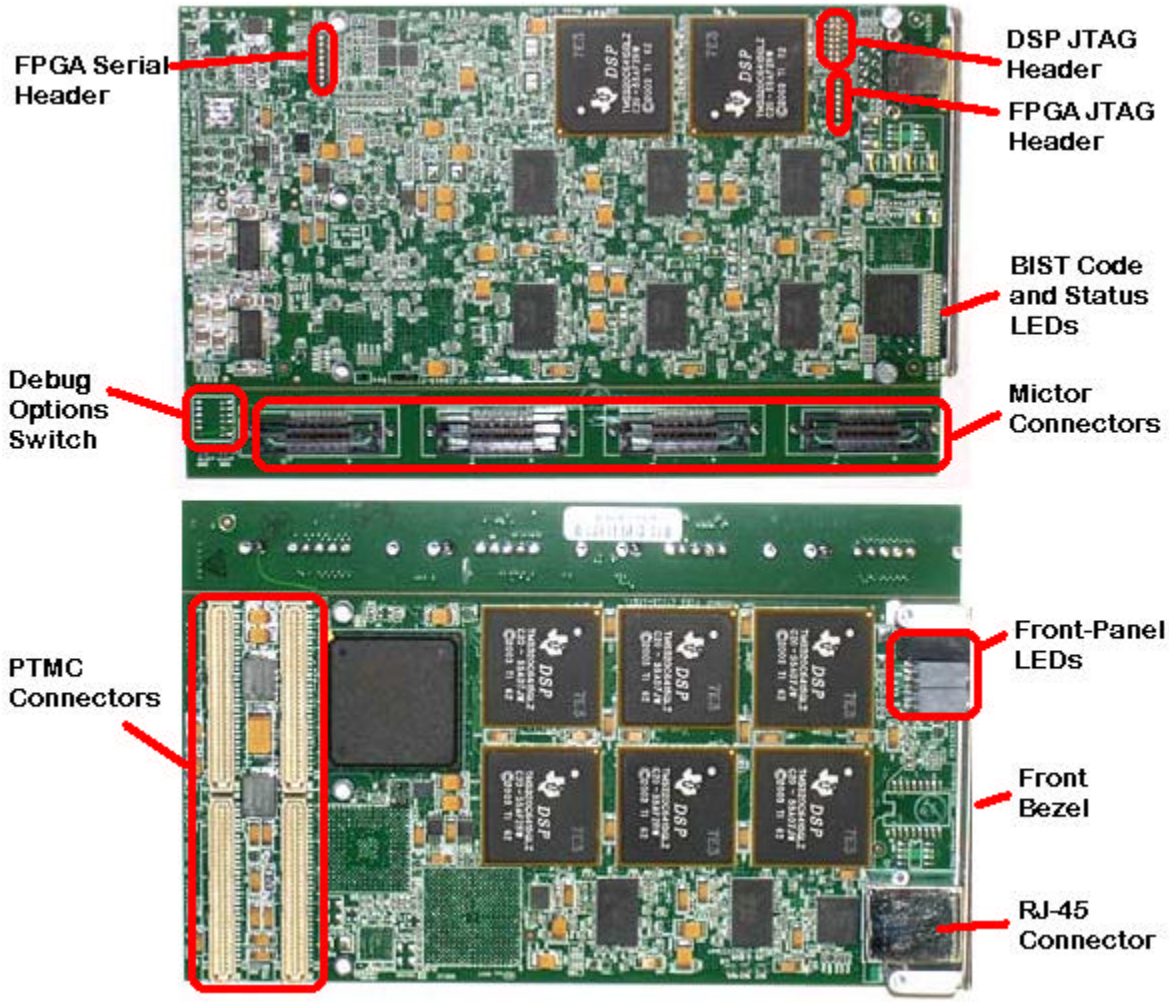


Figure 3-9 SigC6415-PTMC Card, with Headers and Connectors Indicated

3.2.1 Installing PTMC Cards in a PC

When installing PTMC cards in a PC, a standard 33 MHz / 32-bit PCI slot must be available and a PMC-to-PCI adapter card must be used. If the motherboard provides 3.3V signaling, then a "passive only" adapter card can be used, otherwise an active adapter card with 5V-to-3.3V level translation – normally in the form of a transparent bridge device or translation bus switch devices – must be used.

Figure 3-10 below shows “before and after” images of a Rev 1.0 SigC6415-PTMC card connected to a Dynamic Engineering “PCIBPMC” adapter card with bridge device and active voltage level translation. An active adapter allows the SigC6415 board to be inserted into either a 5V or 3.3V PCI slot in a standard PC.

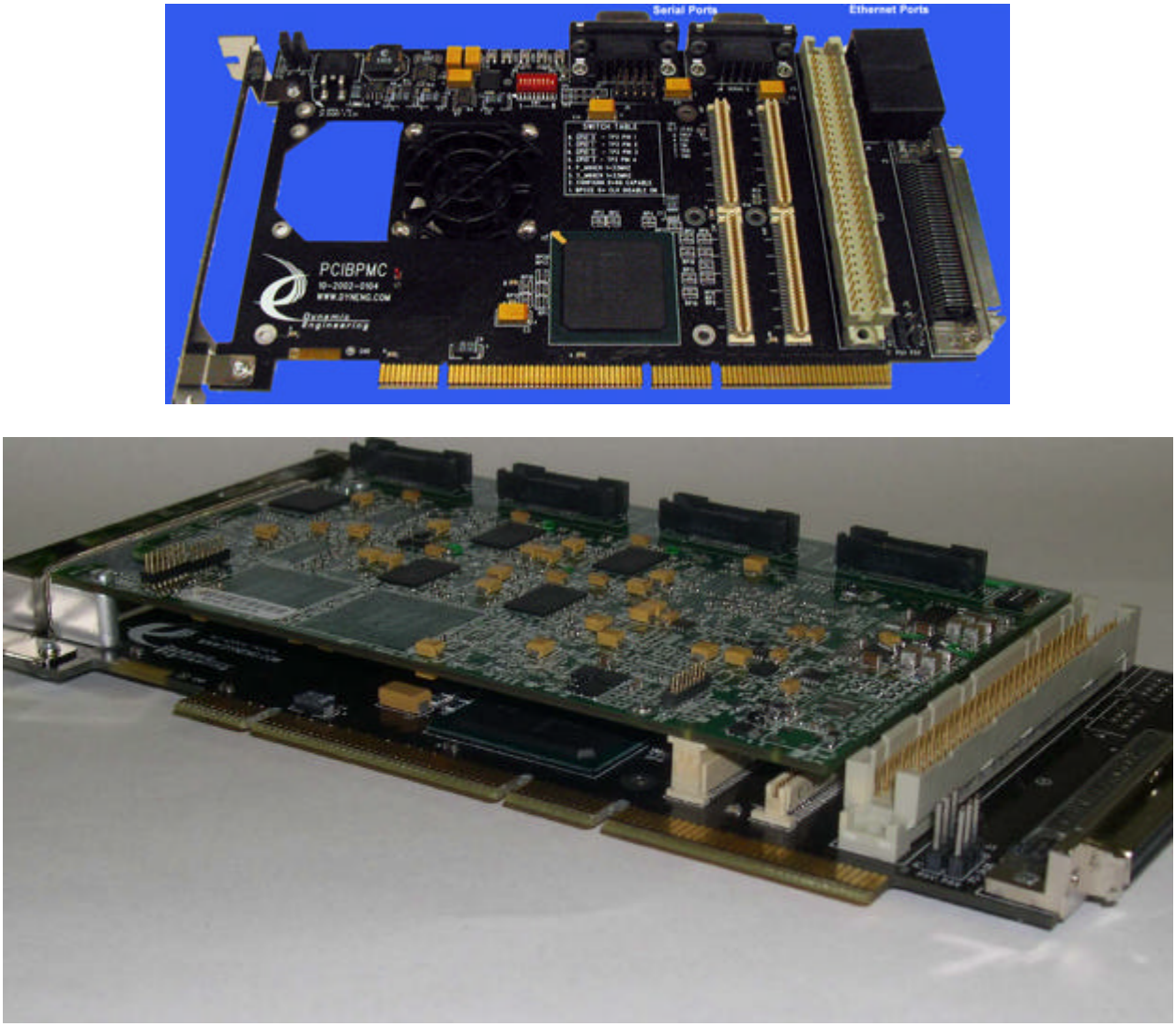


Figure 3-10 SigC64xx-PTMC Card Connected to Active PMC-to-PCI Adapter Card, with Voltage Level Translation

Figure 3-11 below shows a “before and after” images of Rev 1.0 SigC6415-PTMC card connected to a passive PMC-to-PCI adapter card, with no voltage level translation. A passive adapter limits the SigC6415 board to insertion in a 3.3V-only PCI slot in a standard PC.

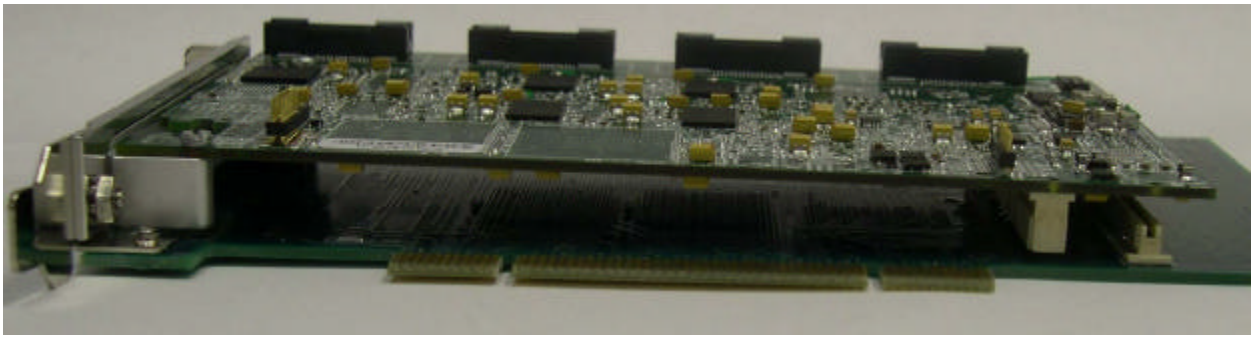


Figure 3-11 SigC64xx-PTMC Card Connected to Passive PMC-to-PCI Adapter Card

3.2.2 Installing PTMC Cards on a cPCI Carrier Board

Figure 3-12 below shows a SigC6415-PTMC card installed on an HW400c/M cPCI carrier board.

Figure 3-12, SigC6415-PTMC Card Installed on HW400c/M cPCI Carrier Board

In Figure 3-12 above, the SigC6415-PTMC card is installed on the HW400c/M board in PTMC site #1. Note that if the debug Tab is attached to the board, then PTMC site #1 must be used to allow the cPCI board to insert into a standard cPCI chassis slot, otherwise either PTMC site may be used.

 Copyright © Signalogic 2003-2005

3.2.3 PTMC Card Installation Steps

Following are important hardware installation steps that should be performed and verified prior to applying power and running the system.

Figure 3-13, Debug Tab, Configuration Switch and Headers

 Copyright © Signalogic 2003-2005

Figure 3-14, DSP JTAG Connection

Note in figure 3-14 above, a “translator adapter” is used to allow the 14-pin 0.1” pitch JTAG emulator connector to fit the small-pitch header used on the SigC6415-PTMC card. If you need this type of “translator adapter”, please contact Signalogic engineering staff.

 Copyright © Signalogic 2003-2005

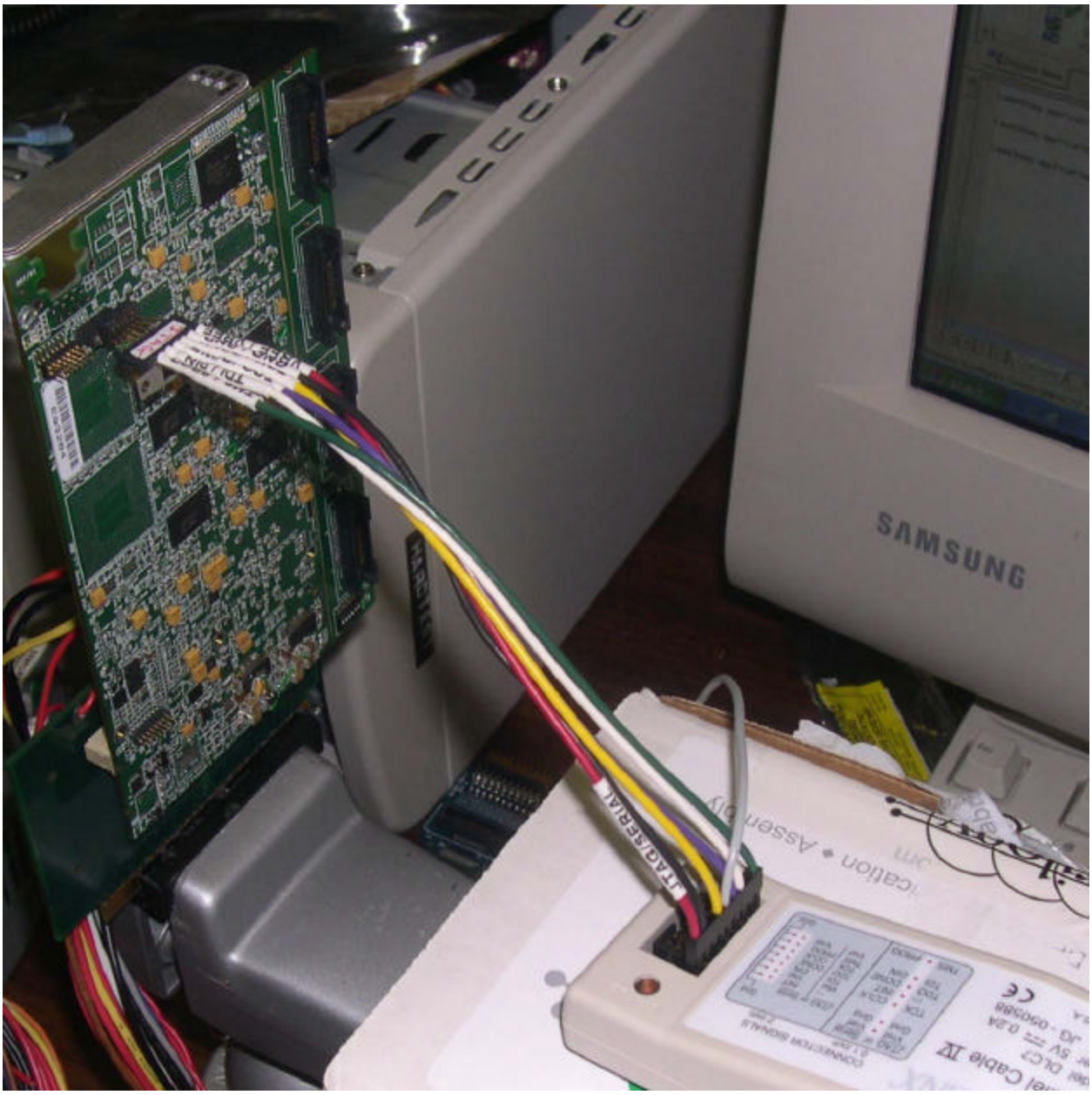


Figure 3-15, FPGA JTAG Connection

Note in figure 3-14 above, a “translator adapter” is used to allow the fly leads from the Xilinx “Parallel IV” pod to connect to the small-pitch header used on the SigC6415-PTMC card. If you need this type of “translator adapter”, please contact Signalogic engineering staff.

 Copyright © Signalogic 2003-2005

1. Configuration Switches

If your SigC6415-PTMC board has the debug tab attached, then SW1 is installed and boot configurations may be set as shown in the table below:

SW1 Switch	Description	Options	Config R
SW1-1	PTMC JTAG Access	On = Enabled	R35
SW1-2 – SW1-3	JTAG Chain Selection	00 = undefined 01 = 10 = 11 = FPGA JTAG chain selected	R33, R31
SW1-4 – SW1-6	PTID Code	000 = "PMC Operation" (PN3 and PN4 disabled) 001 = PTMC Config 1 010 = PTMC Config 2 011 = PTMC Config 3 100 = PTMC Config 4 (not supported) 101 = PTMC Config 5 110 = PTMC Config 6 (not supported) 111 = PTMC Config 7 (not supported)	R29, R27, R25
SW1-7 – SW1-8 **(Note: not yet implemented)**	DSP Boot Source	00 = host software 01 = onboard NOR Flash 10 = 10/100 RJ-45 11 = undefined	N/A

If the debug tab is not attached to the board, then the above options may still be set, using the pull-down Rs listed in the "Config R" column at far right. A pull-down R installed is equivalent to "on" position of SW1.

When SW1-7/8 is set for DSP boot from onboard Flash EEPROM, the following actions are performed:

1. Automatic DSP type detection (e.g. C6414, C6415, C6416)
2. DSP processors are removed from Reset, but not run (idle mode)
3. Onboard Flash EEPROM contents are checked for a valid "executable code" signature; if found, code is bootloaded into one or more DSPs and DSPs are placed into run mode

SW1-7/8 should be set to "00" if the board is being controlled by a host processor in the system, and/or DSP code is being downloaded by a host processor (this is the default mode). However, in that case, before attempting to run CCS software, a test program, host program, or other host tool should be used to take the DSP processor(s) out of reset before invoking CCS.

For more information about EEPROM programming and run-time boot, see sections 10.7.2 and 10.7.3 below, EEPROM Programming Instructions and EEPROM Boot Flow.

2. Optional, if Using JTAG Emulator If you are using a JTAG emulator, then you should connect to J2 header on the PTMC card, as shown in Figure 3-14 above.

Note also that the JTAG connector is keyed! Pin 6 is removed on J2 to match the keyed header provided by most JTAG emulators that support Texas Instruments DSP hardware. *Do not force the connector and try to connect the header in reverse!*

3. Final Safety Checks When the card is correctly configured, then perform the following crucial last-minute checks:

- *Ensure PTMC connector pins are correctly inserted!! Look very closely. There should be no pins off by one row, or outside the connector*

- *Make sure the card is well-seated and pushed down firmly. The card should not be angled, or rock back and forth.*

It must be emphasized that if you plug the card in wrong it will likely be damaged due to a hideous and brutal condition, such as +12V going to a 3.3V IC pin, that will kill a logic device or other IC responsible for bus-interface on the card. In that case, necessary repairs are not covered under warranty!! So please be very careful during card installation – you have been warned.

4. Applying Power Immediately after applying power, you should see the board LEDs labeled CR1 – CR16 illuminate with a BIST (built-in self test) pattern, indicating normal and valid logic boot status. If any of the CR1-CR16 LEDs is flashing then a logic or hardware error condition has occurred.

If you do not see the CR1-CR16 LEDs illuminate in any way, then turn power off without delay and examine the board and carrier card insertion carefully.

3.3 Installing M6x Hardware

For installation of M6x drivers and software, please see the following sections:

- 2.2.4.1.2 above, Installing M6x Board WinXP Drivers
- 2.2.6 above, Hypersignal-Macro Software Board Support

The Hardware Reference Guide for M6x boards is located here:

ftp://ftp.signalogic.com/documentation/Boards/M6x/m6x_hw_manual.pdf

For installation of M6x hardware, please follow the steps below.

For additional information, click on these links:

http://www.signalogic.com/tech_support/m67_support.htm#m62install

http://www.signalogic.com/tech_support/m67_support.htm#m67install

http://www.signalogic.com/tech_support/m67_support.htm#m67issues

3.4 Installing Texas Instruments DSK C54xx and DSK C67xx Boards

Installation of TI DSK boards should proceed completely according to the TI instructions. Once installed, many of the sections in this document apply, including:

- Section 2, Software Installation
- Section 4, Building DSP Application Software
- Section 5, Building Host Application Software
- Section 6, Developing and Debugging DSP Code

Note however that sections specific to SigC5xxx and SigC6xxx hardware do not apply, including section 3, and sections 7-11.

Below are examples of selecting Texas Instruments DSK boards from Hypersignal software and the Hardware Manager dialog that appears when running MATLAB, Visual C/C++, and Visual Basic programs that use DirectDSP software.

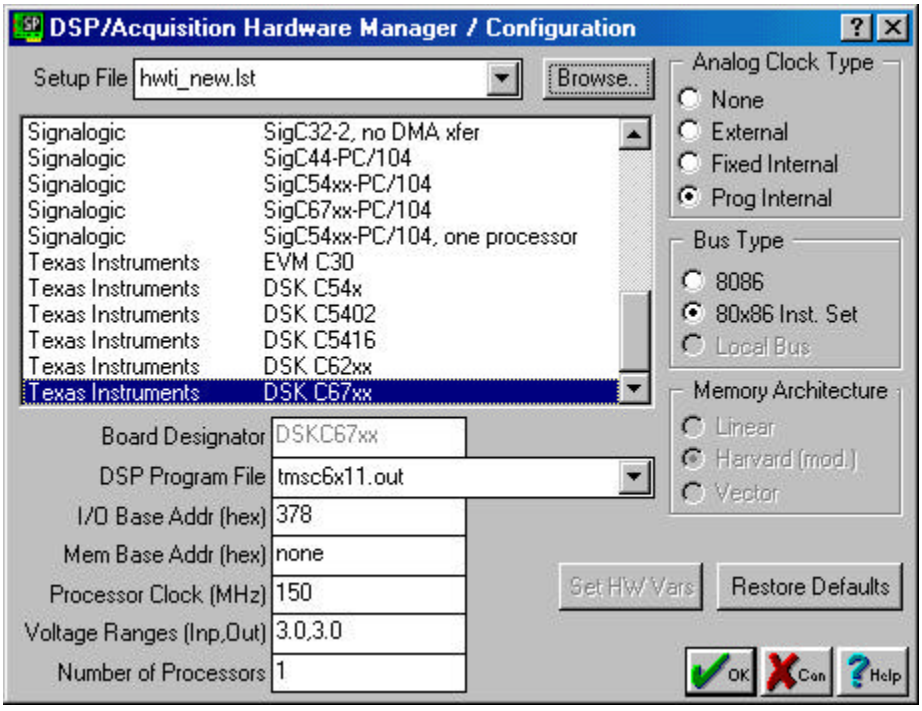


Figure 3-16, Texas Instruments DSK C6711 Board Selected in Hardware Manager Dialog

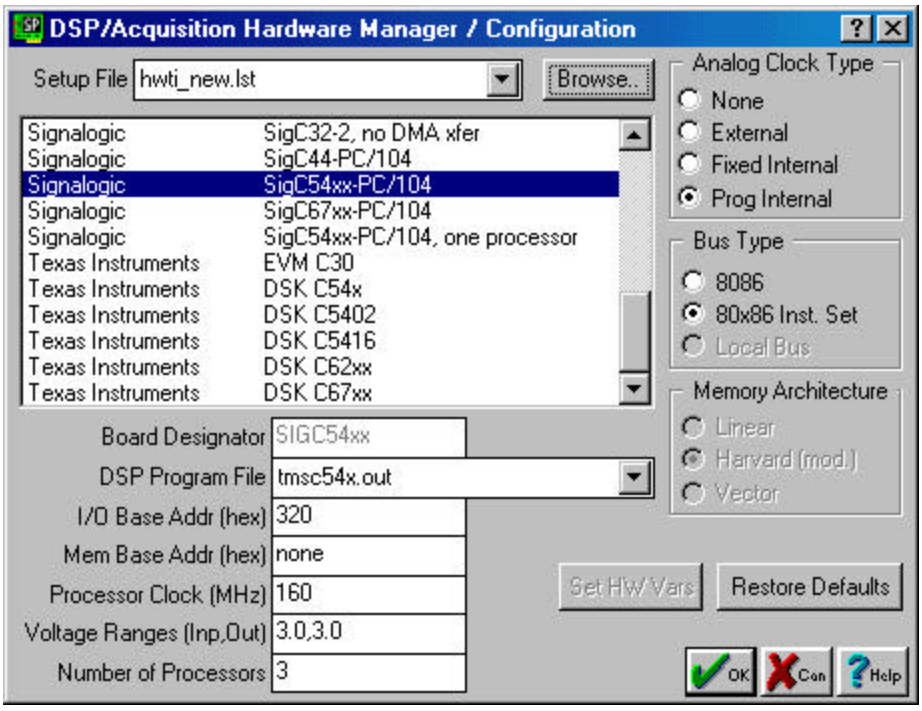


Figure 3-17, Texas Instruments DSK C5402 Board Selected in Hardware Manager Dialog

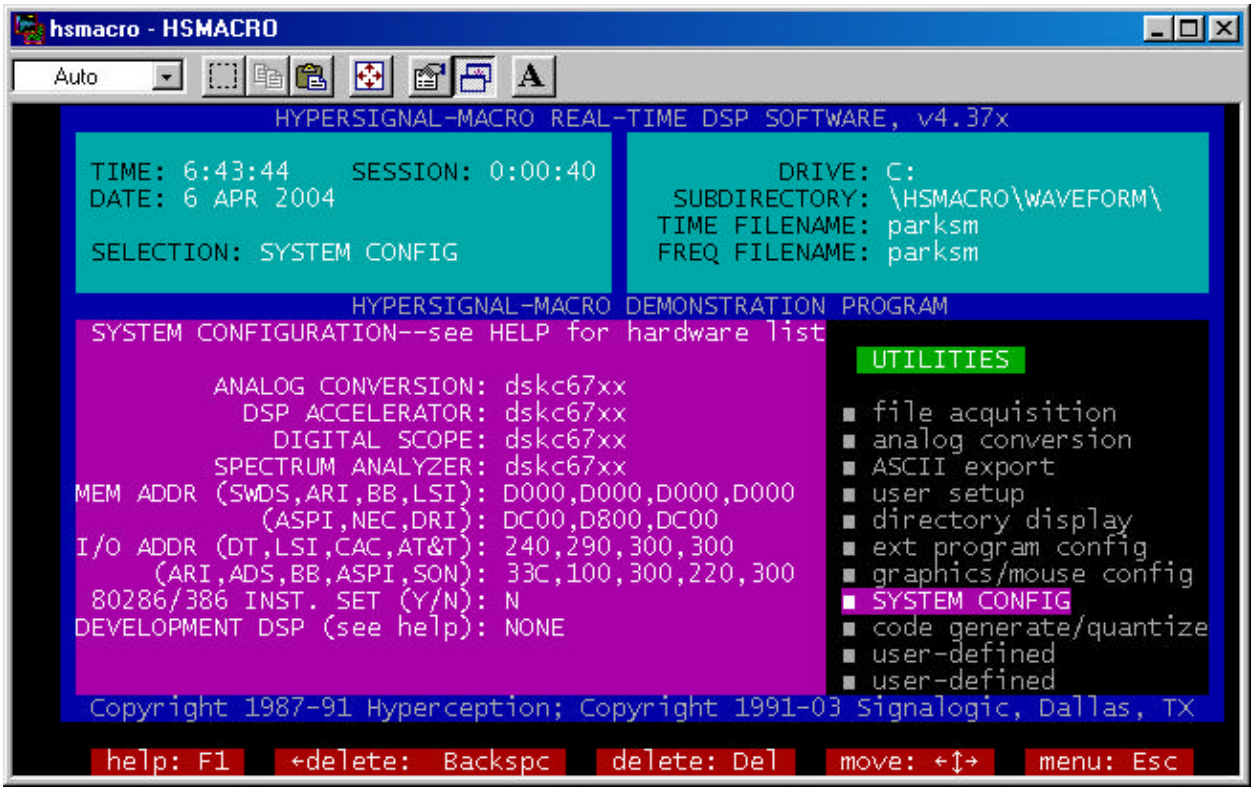


Figure 3-18, Texas Instruments DSK C6711 Board Selected in Hypersignal-Macro System Configuration Menu

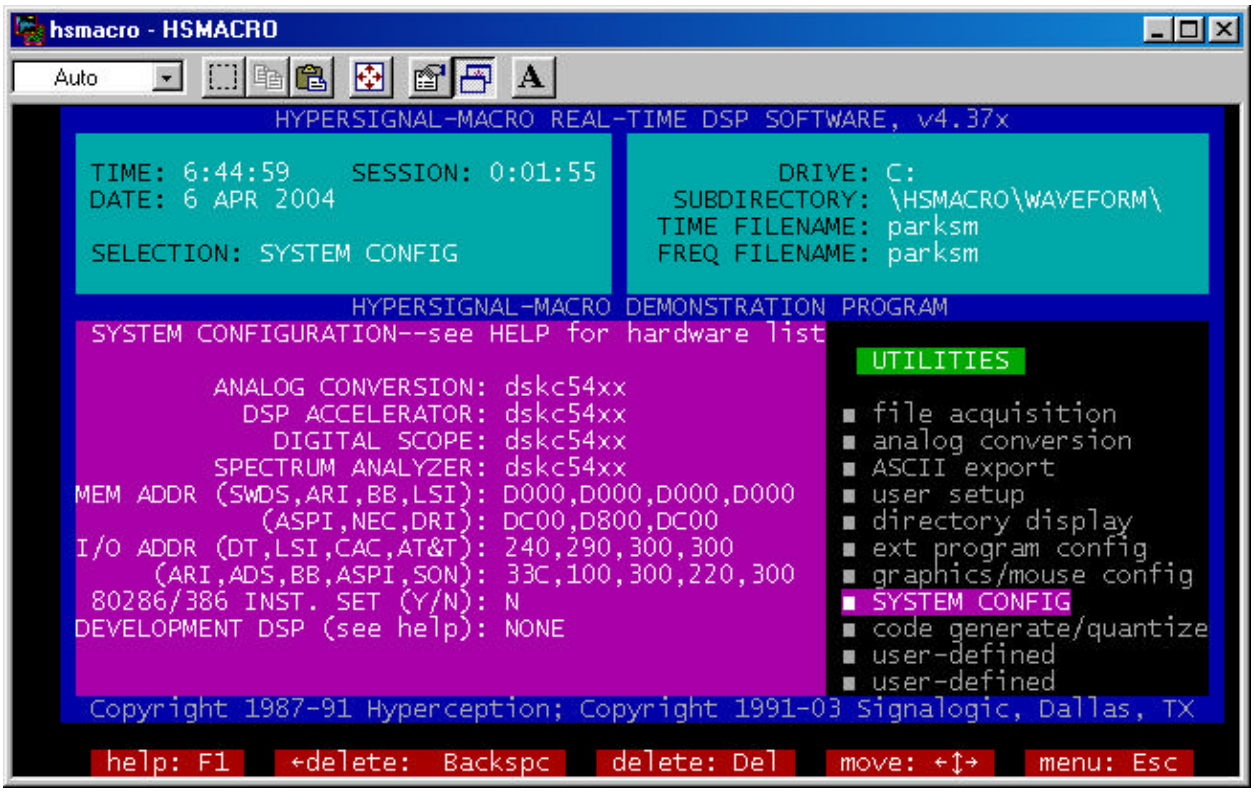


Figure 3-19, Texas Instruments DSK C5402 Board Selected in Hypersignal-Macro System Configuration Menu

 Copyright © Signalogic 2003-2005

3.5 Hardware Test and Check-Out Programs

A series of test programs can be used to test the SigC5xxx and SigC6xxx PC/104 boards. These programs also can be used for diagnostic purposes for individual processor and I/O peripheral resources on the boards.

Below is a summary list of test programs and utilities that can be used to test and check SigC5xxx and SigC6xxx PC/104 and PCI boards, and where in this documentation you can find more information about running the test programs.

Board Resource	**Test Program**	**More Information**
DSP and HPI Interface	board_test.exe	Section 3.5.2.1 and section 10.2
Host Interface Logic	board_test.exe	Section 10.3
DMA Controller	dma_test.exe	Section 3.5.2.2 and section 10.5
DUART	duart_test.exe	Section 3.5.2.3 and section 10.6
Digital I/O	digio_test.exe	Section 10.8

3.5.1 Stand-Alone Mode Test

The standard .out file delivered with Hypersignal or DirectDSP software defaults to a board test mode when stand-alone operation is detected. This is applicable when the .out file has been built with the using the 'Standard' Active Configuration for the C5xxx or C6xxx SCI project (see section 4.1 below, CCS Project Build & Run Sequence).

Note that stand-alone operation" is defined as whenever no host software is detected; i.e. if the DSP is running without being initialized by Hypersignal or application level software that uses DirectDSP, such as MATLAB application, Visual C/C++ or Visual Basic GUI, or other user-defined GUI. Stand-alone vs. host mode detection is automatic, performed by DSP low-level boot code at run-time.

There are two (2) ways that stand-alone operation is typically initiated:

1) Download executable DSP code using Code Composer Studio (CCS) software, via JTAG header on the board. Normally, a JTAG emulator such as Spectrum Digital or SEED is used

2) Boot from Flash EEPROM (requires JP14 to be installed, and valid program to be contained in EEPROM -- see section 10.7 below, Boot/Flash EEPROM)

When the default stand-alone test mode is operating, default source code in main.c is activated which performs:

- Analog I/O loopback (19.2 kHz sampling rate, buffer size = 300)
- RS-232 loopback on UART0 (19.2 kbaud, 8-bit word, 1 stop, no parity)
- Digital I/O loopback on bank 2 (first 4 bits are input, second 4 bits are output)
- Flashing LEDs (2-bit value increments from 0 to 3 and repeats)

Note that rate of LED flashing is fast and it may almost look like both LEDs are constantly on; actually both LEDs are changing, from both off, to 1 on, to the other on, to both on. The rate of LED flashing can be adjusted by changing the “main.c” file in C5xxx or C6xxx SCI software.

3.5.1.1 CCS Startup and Stand-Alone Mode Test

The standard sequence for running Code Composer Studio (CCS) and test/example DSP code is given below. This assumes you have CCS already installed and functioning correctly – for more information on this subject, please see section 2.1 above.

1) Follow instructions in section 3.0 above, Hardware Installation, to ensure the system and board are installed and setup correctly. If you have a PC104 board, ensure the board's rotary switch setting for I/O base address is set correctly; see section 3.1 above. If you have a PTMC board, see section 3.2.

2) Boot PC with JTAG emulator connected. Recommended JTAG emulators include Spectrum Digital XDS510PP and XDS510PP-Plus, and SEED XDSpp. The emulator should be connected to the parallel port on the PC, and the parallel port should in EPP mode in the PC’s BIOS setup. See section 2.1.5 above, Installing JTAG Emulator Software. See the hardware installation sections for pictures showing how to connect JTAG emulators to the DSP hardware.

3) Run SDConfig, click on 'R' toolbar button (JTAG reset), click on "emu" toolbar button (JTAG scan test); scan test should pass and show:

```
Found JTAG IR length of 8
Found 1 JTAG device(s) in the scan chain
```

Note that the SDConfig utility is installed with the XDS510PP unit software, or can be downloaded directly from http://spectrumdigital.com. Below are some examples of SDConfig screen captures for different DSP hardware types:

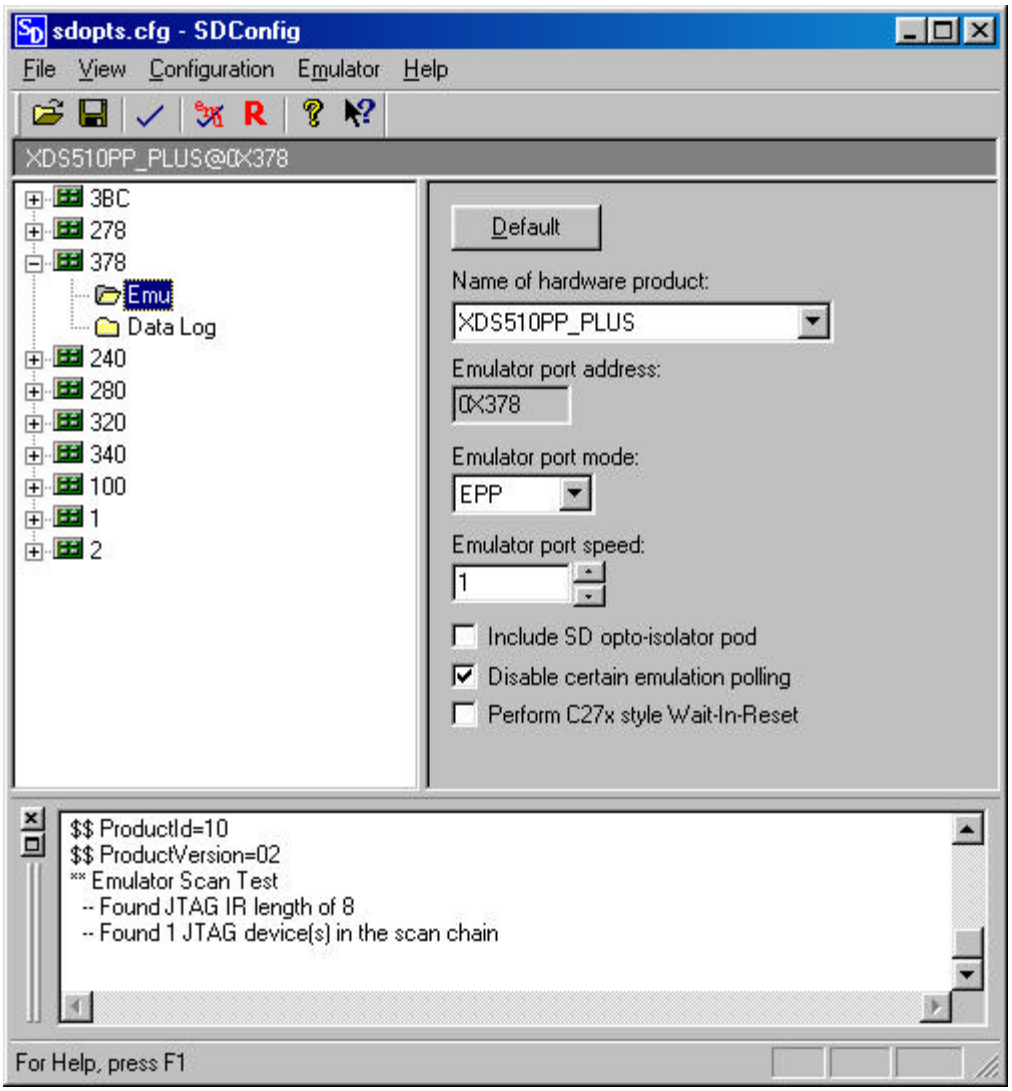


Figure 3-20, Example SDConfig Display for Single-Processor SigC54xx Board

 Copyright © Signalogic 2003-2005

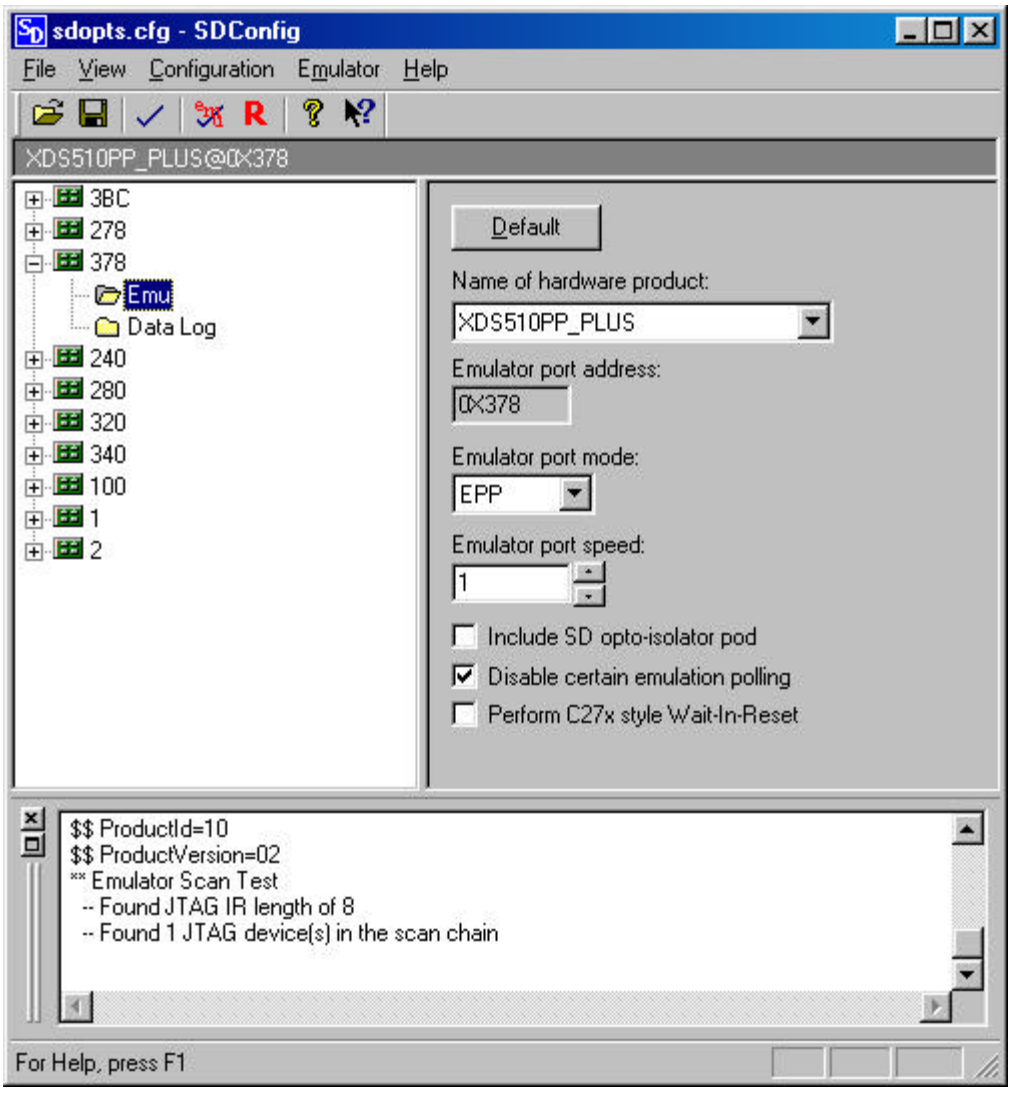


Figure 3-21, Example SDConfig Display for Single-Processor SigC67xx Board

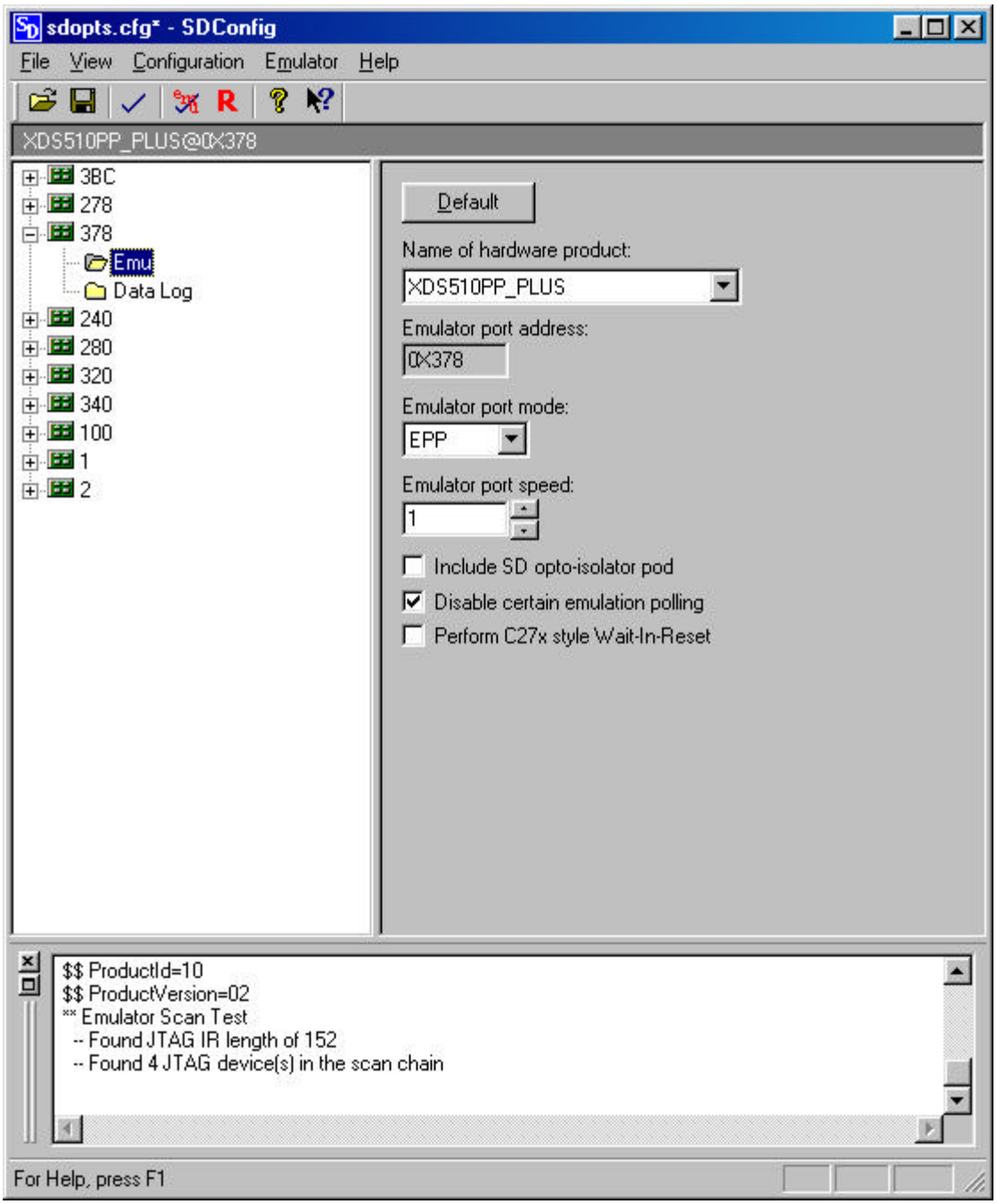


Figure 3-22, Example SDConfig Display for 4-Processor SigC64xx Board

Note that for multi-processor modules, the IR length and device values above should be multiplied by the number of DSP devices on the module.

5) **PC104 board only**. If the PC104 board is in a host PC (using passive carrier card) and stand-alone jumper J14 is not installed, then run "board_test" or "isatest" host test program. Enter the following sequence at the prompts:

2 board type: 2 = PC/104 board type, 3 = PCI
2 processor type: 2 = C5xxx, 3 = C6xxx
1 module site: determines which module site to test
1 number of processors: number of processors on the module
12 test type: typical examples include 2 for core select, 3 for register-level and low-level onchip memory test, and 12 for CCS preparation

Test 12 releases DSP(s) on the SigC5xxx board from reset. Note also that other test programs could be used at this step, including Hypersignal-Macro software functions

If the PC104 board is not installed in a PC, or the stand-alone jumper J14 is installed, then skip the above step. In this case the onboard logic will automatically release DSP(s) from reset at power-up.

6) Run CCS software.

7) Load DSP executable (binary) file, as follows:

File | Load Program

Select desired .out file, for example tmsc64xx.out under \dspcode\c6xxx\SCI\cStandard

8) Perform menu command Debug | Run or Debug | Run Free.

For detailed information experimenting with the stand-alone mode test C source code, for example editing main.c and rebuilding in CCS, please see section 4 below, Building DSP Application Software.

3.5.1.2 CCS Startup and Stand-Alone Mode Test for MELPe

The standard sequence for running the MELPe test/example code is as given in section 3.5.1.1 above. Additional, MELPe-specific instructions are provided below:

1) After running CCS software (step 6 in section 3.5.1.1 above), load MELPe executable (binary) file, as follows:

File | Load Program

Select melpe.out under \dspcode\c5xxx\MELPe\cStandard

2) Perform menu command Debug | Run or Debug | Run Free.

3) At this point the DSP code should be running full-duplex MELPe. For example, the following are default settings of property values in the high-level MELPe C source code (melp_rt.c):

opmode	REAL_TIME
audio_loopback	0

melpmode	ANA_SYN
wFramesize	180
sync_serial_enable	SS_PACKET_TRANSFER_MODE
lBaudRate	2400

With the above default values, the test code should run one-channel 8 kHz audio I/O on McBSP0, full-duplex MELPe, and 2400 bps RS-422 transmission on McBSP1. If audio input is connected to channel 0 and a loopback plug connector is installed on RS-422 port A (connector marked J30 on PC/104 board, connect pins 2-3), then you should hear MELPe compressed/decompressed on channel 0 audio output.

The loopback connector basically loops back the Tx (Transmit line) to Rx line (Receive line) of the Synchronous serial port; in this mode clock and framesync are generated by the onboard Routing Logic peripheral and Tx framesync is tied to Rx framesync within the logic, as are Tx clock and Rx clock. For a picture of a loopback connector attached to RS-422 port A, see the picture in Figure 3-8 above.

4) At the point where you wish to make changes to the high-level C interface (melp_rt.c file), you can load the MELPe project, edit source code files, and rebuild. After loading the MELPe project, perform the following steps:

Select “Standard” configuration; this is the default active configuration after loading the project (note: do not attempt to build or use the TI_DSP_BIOS configuration, as it is in a preliminary stage)

After successful build, go to step 1) above, reload the melpe.out file, and re-run.

Note that additional instructions about improving C code real-time performance are given in section 4.1.6 below, Application Code Performance and Architecture Consierations.

Additional MELPe Demo Code Notes

1) The MELPe demo code has been tested on a 160 MHz C5416 board. That means onchip data memory available at 0x8000, so the demo may not run correctly on C5409A for which data mem at 0x8000 is located offchip.

3.5.1.3 Flash EEPROM Boot and Stand-Alone Mode Test

3.5.2 Board Peripheral Testing

3.5.2.1 Basic DSP HPI Testing

 Copyright © Signalogic 2003-2005

3.5.2.2 DMA Controller Testing

3.5.2.3 Flash EEPROM Testing

3.5.3 Linux Test Programs

The sections below give instructions for running Linux test programs.

3.5.3.1 Running Board and Memory Test Programs

Test programs are provided with DirectDSP software to both test SigC5xxx and SigC6xxx boards, and also to serve as example source code.

To run one of the test programs provided with DirectDSP software called “boardTest”, follow the steps shown in the screen capture below.

Note that all instructions in this section assume the correct LKM driver has been loaded; see section 2.2.5 above, Linux Driver Install Instructions.

10.0.0.7 - PuTTY
EN English (United States)
[root@blackhawk2 boardTest]# ./boardTest -h
Command line option syntax:
-b |integer|: Set IO base address in Hex (Only for PC104 boards), (e.g. -b320).
-f |integer|: Set processor clock rate in MHz (e.g. -f720).
-o |string|: DSP COFF file name (e.g. -osigc6415.out|.
-p |integer|: Set number of processors on board (e.g. -p1).
-s |string|: Board designator. (e.g. -sSIGC64XX).
-t |boolean|: Run with talker enabled.
-h |boolean|: Display this help list.
[root@blackhawk2 boardTest]#
[root@blackhawk2 boardTest]# ./boardTest -p1 -f720 -oc64timer.out -sSIGC64XX

Figure 3-23, Running boardTest Program

Note that the above capture shows command-line options followed by example run command. When running the board test program, you should see the following output, on the system display defined as “stdout”:

```
userIfs = {1, 720, 0, c64timer.out, SIGC64X, 0}
boardStr = SIGC64X
IObaseAddr = 0
DSP COFF file = c64timer.out
Initializing board...
before assign board
m_hBoard = 0x6b8b4567
m_szBoardsf = SIGC64X
m_wIOBaseAddr = 0x0
m_nProcList=1, m_nMaxProcessors=1
DSInitProcessor: wProcessorList = 1
SlotIndx = 0,
Board_Type 0x8806
Talker is not used.
DSXInitBoard C64x!
Inside InitBoardT50
ds64xInitBoard: Read REG_EXTx_SEL passed.

ds64xInitBoard: passed.
DSXInitBoard Successful!
board class 2054
```

```
  { NOTE:  COFF file download data displayed here if –v (verbose) option enabled }


Starting real-time DSP code...
begin
0x502a  0x502a  0x502a  0x502a  0x502a  finish
 csl regi = 0x1
reset regi = 0xff
regi 0 = 0x12502a
DSXRunBoard Successful!
RunBoard: regi 0 = 0x12502a
Hardware Running

_memBuf = 0x2a40

begin
 0x00000000 0x00000001 0x00000002 0x00000003 0x00000004 0x00000005 0x00000006 0x00000007
0x00000008 0x00000001 0x0000000a 0x0000000b 0x0000000c 0x0000000d 0x00002078 0x0000000f
0x00000010 0x00000011 0x00000012 0x00000013 0x00000014 0x00000015 0x00000016 0x00000017
0x00000018 0x00000019 0x0000001a 0x0000001b 0x0000001c 0x0000001d 0x0000001e 0x0000001f
0x00000020 0x00000021 0x00000022 0x00000023 0x00000024 0x00000025 0x00000026 0x00000027
0x00000028 0x00000029 0x0000002a 0x0000002b 0x0000002c 0x0000002d 0x0000002e 0x0000002f
0x00000030 0x00000031 0x00000032 0x00000033 0x00000034 0x00000104 0x00000036 0x00000037
0x00000038 0x00000100 0x0000003a 0x0000003b 0x0000003c 0x0000003d 0x0000003e 0x0000003f
_memBuf = 0x2a40

begin
 0x00000000 0x00000001 0x00000002 0x00000003 0x00000004 0x00000014 0x00000006 0x00000007
0x00000008 0x00000020 0x0000000a 0x00000000 0x0000000c 0x0000000d 0x0000000e 0x0000000f
0x00000010 0x00000011 0x00000012 0x00000013 0x00000014 0x00000015 0x00000016 0x00000017
0x00000018 0x00000019 0x0000001a 0x0000001b 0x0000001c 0x0000001d 0x0000001e 0x0000001f
0x00000020 0x00000021 0x00000022 0x00000023 0x00000024 0x00000025 0x00000026 0x00000027
0x00000028 0x00000029 0x0000002a 0x0000002b 0x0000002c 0x0000002d 0x0000002e 0x0000002f
0x00000030 0x00000031 0x00000032 0x00000033 0x00000034 0x00000004 0x00000036 0x00000037
0x00000038 0x00000039 0x0000003a 0x0000003b 0x0000003c 0x0000003d 0x00000026 0x0000003f
_memBuf = 0x2a40

  { NOTE:  additional buffer print-outs continue here... }
```

Figure 3-24, Board Test Program Output

Note that the above output is also logged to file; look for file "log" under

```
../apps/boardTest/
```

in the DirectDSP software installation directory structure.

3.5.3.2 Running Data Scope Test Program

There are two (2) versions of the Data Scope test program provided for Linux systems:

- Command-line, with text only display. Suitable for console or terminal-based interface
- GUI version, with graphical waveform display. Requires wxGTK to be installed

To run the command-line Data Scope test program, perform the steps shown in the screen capture below:

Figure 3-25, Running Data Scope Command-Line Test Program

You should then see the following screen display

Figure 3-26, Data Scope Command-Line Test Program Display

To run the GUI version of the Data Scope test program, perform the steps shown in the screen capture below:

Figure 3-27, Running Data Scope GUI Test Program

You should then see one of the following Hardware Manager dialog displays:

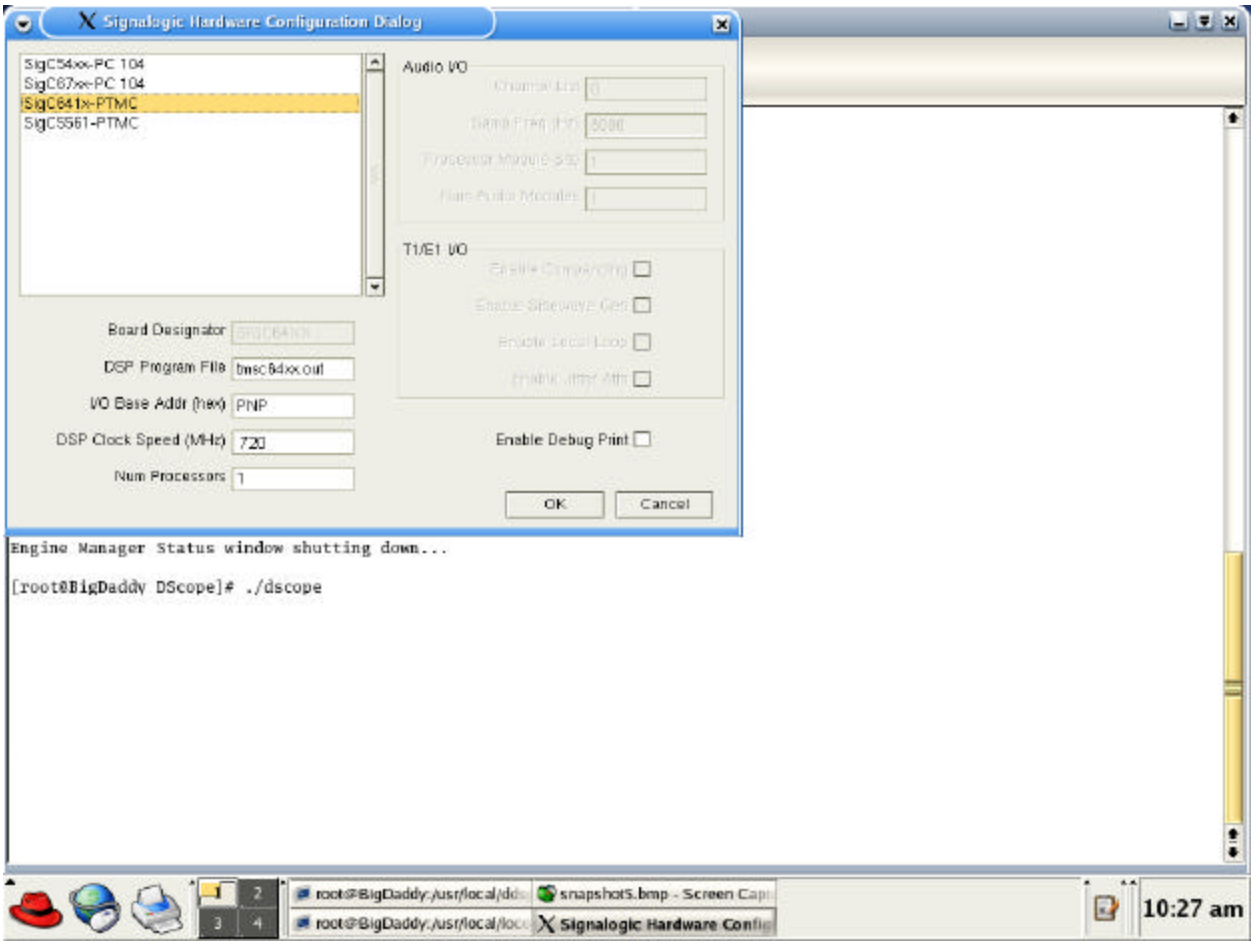


Figure 3-28, Data Scope GUI Test Program, Hardware Manager display with SigC641x-PTMC board selected

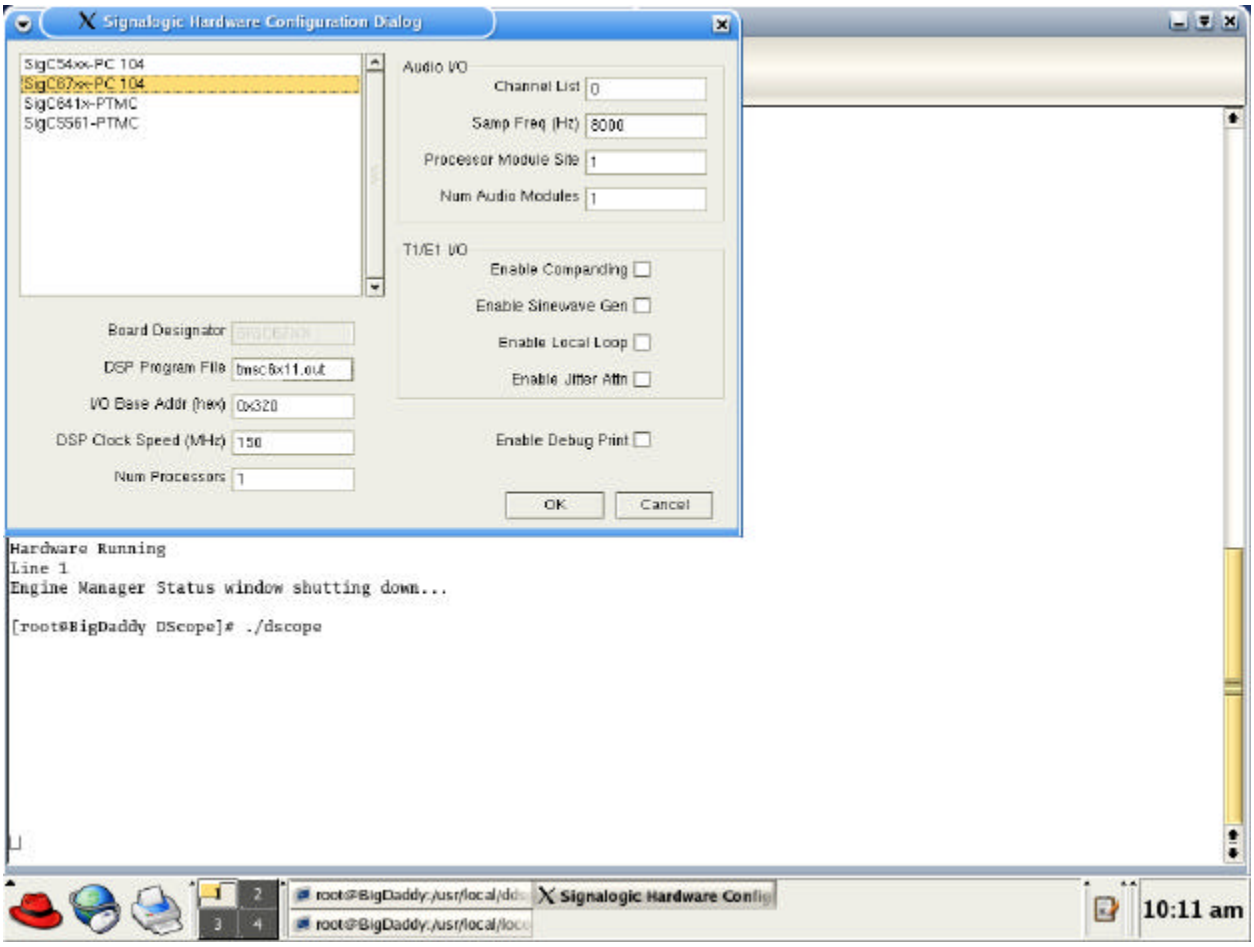


Figure 3-29, Data Scope GUI Test Program, Hardware Manager display with SigC67xx-PC104 board selected

After making appropriate Hardware Manager dialog selections, click on the Ok button. You should then see the following display:

Figure 3-30, Data Scope GUI Test Program Display

3.5.3.3 Running the E1-Audio Test Program

***NOTE**: The following instructions assume that the SigFM.o module has been built from the SigFModule directory, and that libEngMgr.so has been built from the EngMgr directory. If not, see software build instructions in section 5.3 below, Building Linux Software.*

After initial system reboot, the kernel module must be loaded. See instructions in section 2.2.5 above, Linux Driver Install Instructions.

 Copyright © Signalogic 2003-2005

To run the E1-Audio test program, first enter:

```
shell/> startx
```

This will start X Windows, a Linux graphical user interface. After X Windows starts up, holding down the left mouse button (or right, depending on which window manager program is running) will bring up a menu. Choose "XTerm" from the menu. This will bring up a small window that represents a console. The following commands must be typed in from this console window:

```
shell/> /sbin/insmod SigFM
```

This will install the SigFM.o module into the kernel so that the Signalogic software can run. Note that XTerm is an option only under X Windows, and not KDE, which is the default window manager provided with Red Hat Linux. On systems with relatively small amount of memory, such as the Lippert CoolRunner II, KDE may not be used due to memory considerations.

Then move to the subdirectory from which the E1-Audio test program will be run, as follows:

```
shell/> cd /usr/local/share/HwMgr
```

To run the Signalogic E1-Audio test program, type in the XTerm window:

```
shell/Hwmgr> ./e1test
```

This will run the E1-Audio version of XDScope, one of the test programs provided with the Linux version of DirectDSP software. If e1test.o runs successfully, the Hardware Manager dialog box will appear as shown in Figure 3-31 below:

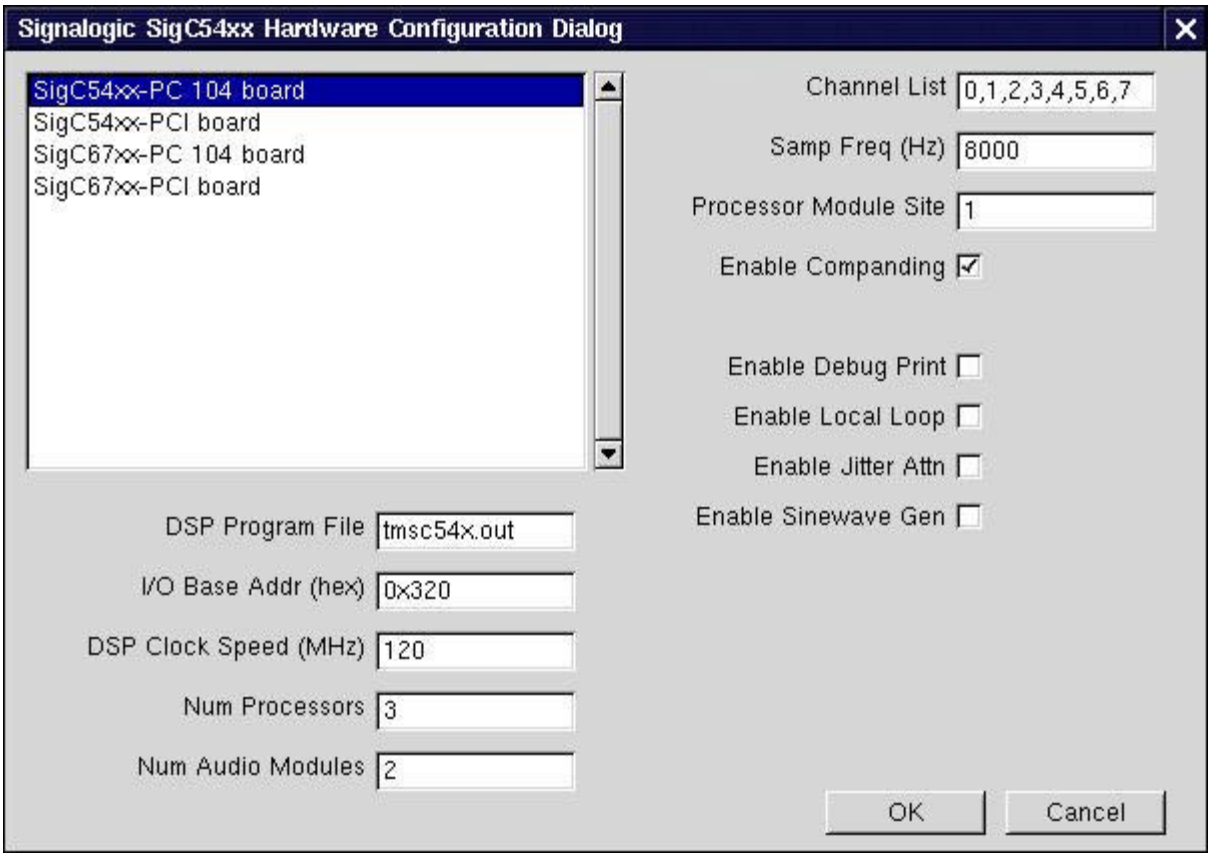


Figure 3-31, Hardware Manager Dialog

Note that in Figure 3-34 below, the XTerm window is also visible, showing the command-line entry used to invoke the E1-Audio test program (e1test.o). Figure 3-34 also includes some of the background screen is shown, with displays visible from XTerm and other steps preceding Hardware Manager dialog display.

Below are some notes about particular fields in the Hardware Manager dialog box:

- DSP program file should be set to its default value of tmsc54x.out
- default base I/O address for the SigC5xxx-PC/104 board is 0x320, but should be set to 0x340 when installed in the Lippert system
- Processor Clock Speed field should be set to:

  100 MHz for C5409 and C5402 modules

  120 MHz for C5409A module

  160 MHz for C5416 modules

  where the module type refers to DSP module installed on the SigC5xxx-PC/104 board. Note that Processor Clock Speed field values should be entered in MHz, for example:

```
100
120
160
```

- the Number of Processors field should correspond to the number of DSP devices installed on the DSP module. The valid range is 1 to 3. If you are not sure, enter 1 which will cause only the first DSP to be activated.
- the Sampling Rate field should be set to 8000 Hz for T1/E1 use (for non T1/E1 use, valid sampling rate range is 4000 to 48000 Hz)
- the Channel List field should contain the desired channels to be displayed, in order of onscreen trace display from top to bottom, with each channel being separated by a comma. The allowable range for channels is 0-7 and the maximum number of traces is 8 (eight). Some example Channel List field entries follow:

```
0,1,2,3     (first four channels displayed)

0,0,3       (channel 0 displayed on traces 0 and 1,
          channel 3 displayed on trace 2)

1        (channel 1 displayed only)
```

- for E1-audio testing, the "Enable Companding" checkbox should be set. This field will cause the DSP to enable A-law companding on McBSP1 data, which is normally routed to the T1/E1 device on the SigC5xxx-PC/104 board. On the transmit side, data is A-law compressed and bit inverted; on the receive side, Rx data is bit-inverted and A-law expanded.
- to run the E1-Audio test program and cause debug information error and status messages to also appear in the XTerm command window (in addition to standard audio trace display), set the Enable Debug checkbox

  NOTE: *if you check Enable Debug, then your audio trace display may be slow because of the additional information being read from the SigC5xxx DSP module and displayed onscreen (see Figure 3-33 below).*

Click "OK" at lower right of Hardware Manager to begin the E1-Audio test program. You should see a digital scope style display as shown below, with XDScope shown in the window title bar. You may need to wait from 1 to 5 seconds for display traces to become active, depending on the speed of your system.

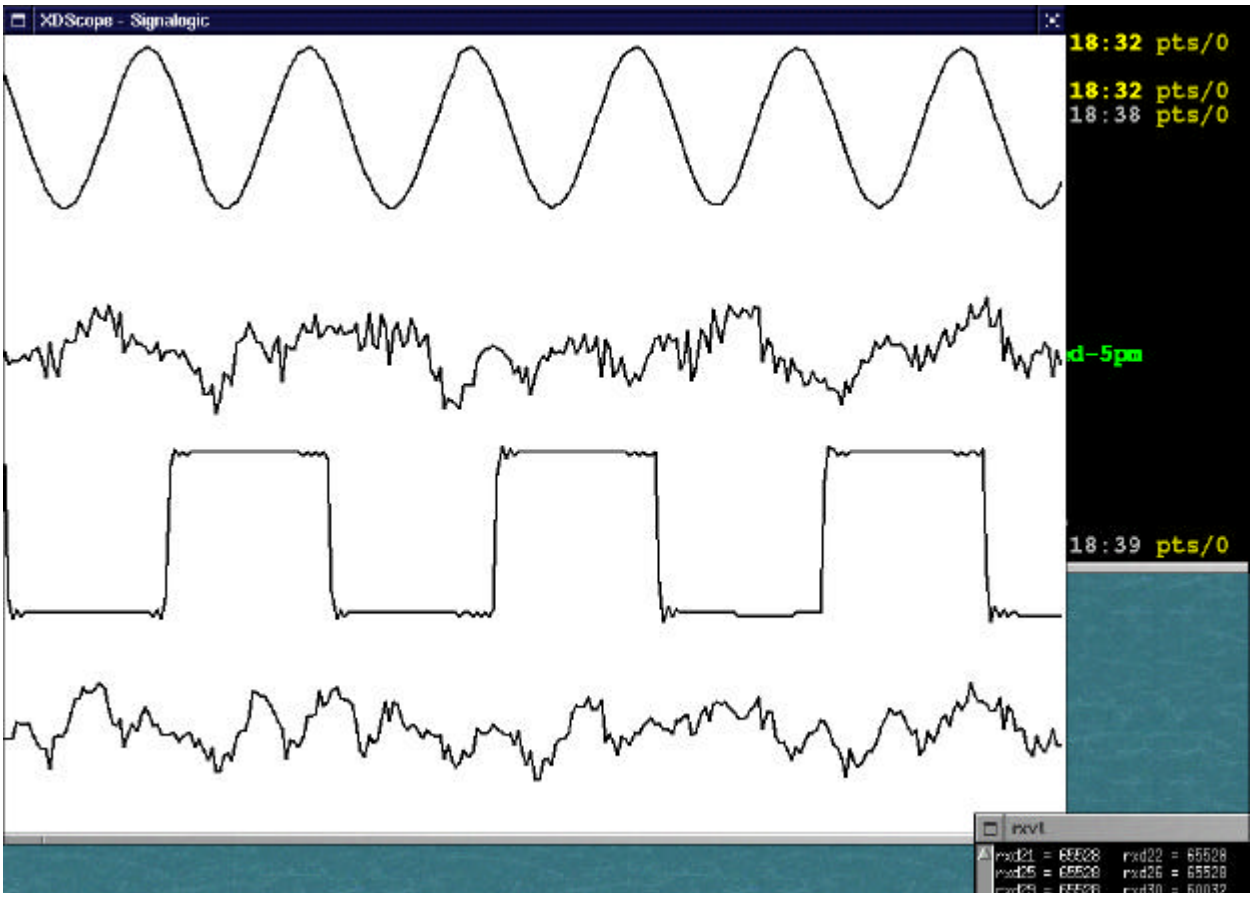


Figure 3-32, XDScope Display of E1 Audio Data

In the above display, test pattern data is shown on traces 1 and 3, and speech data is shown on traces 2 and 4. Up to N data traces received from E1 line input are displayed simultaneously by XDScope software, where N is the number of channels entered in the Channel List field of the Hardware Manager dialog box (see Figure 3-16 above).

Note that for sine wave test data, it may be possible to see the results of A-law companding over E1 channels, in the form of a “stair step” effect at higher signal amplitudes.

In E1 wire-loopback mode (with E1 loopback plug installed in RJ-45 connector), each trace corresponds to the appropriate input channel from the SigSD4 Audio Module. For example, audio input data on SigSD4 input channel 0 will appear (after being transmitted and received over E1) on the first trace, channel 1 on the second trace, etc.

The demo can be exited at any time by pressing "Esc", with the display window in the foreground (i.e. the active window).

The E1-Audio test program performs the following functions:

- initializes the SigC5xxx board, C5xxx DSP processor, and memory-map

- initializes the SigSD4 audio module
- initializes the Dallas Semiconductor DS2155 T1/E1 device
- downloads executable DSP code (COFF file format)
- sets DSP properties that control setup of onboard resources
- runs the DSP, begins audio and E1 processing
- displays continuous buffers of analog input data sent through the E1 circuit
- outputs audio data shown on the display traces

Below is another XDScope screen capture, showing a debug information print out window. To enable debug information display, set the Enable Debug Info checkbox in the Hardware Manager dialog (see Figure 3-16 above).

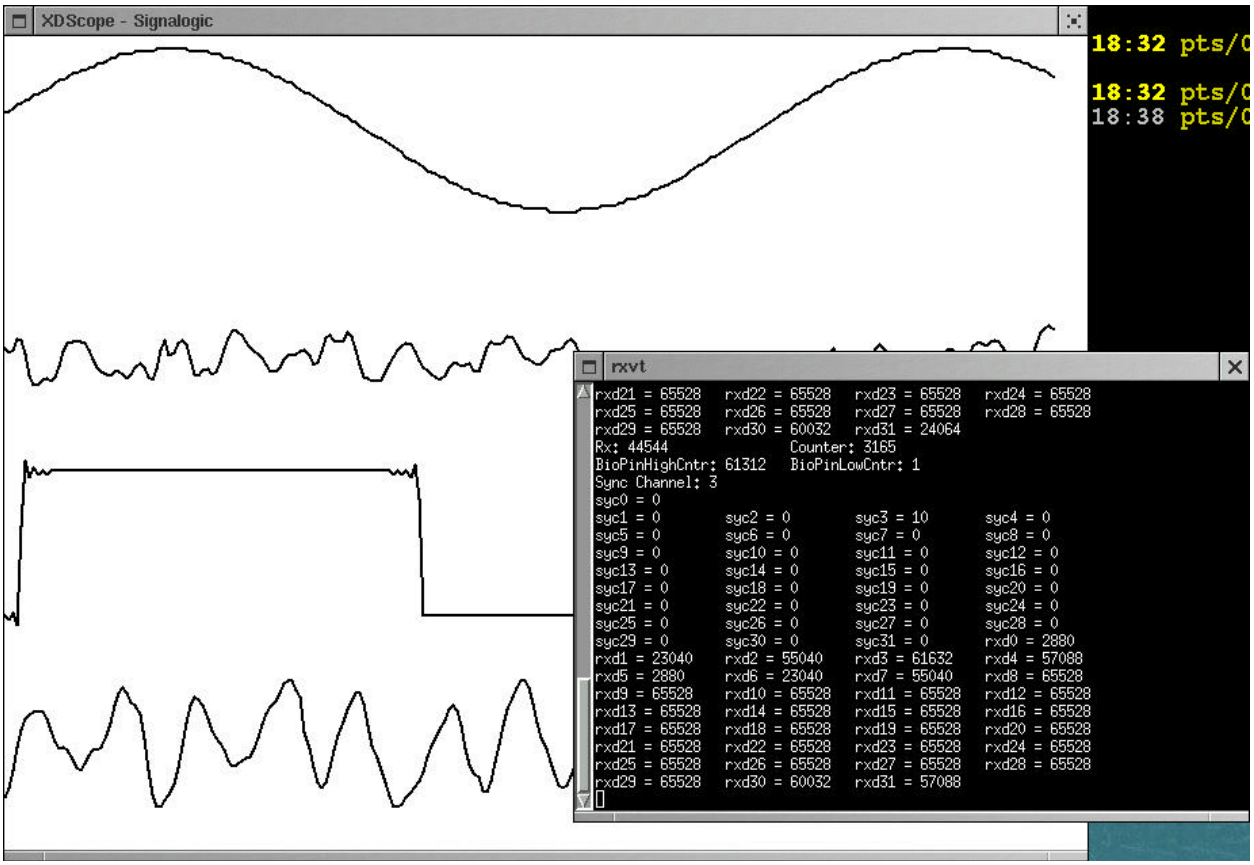


Figure 3-33, XDScope Display with Debug Information Shown

Below is another screen capture of the Hardware Manager dialog, showing some of the XTerm and other software displays that precede running the e1test program.

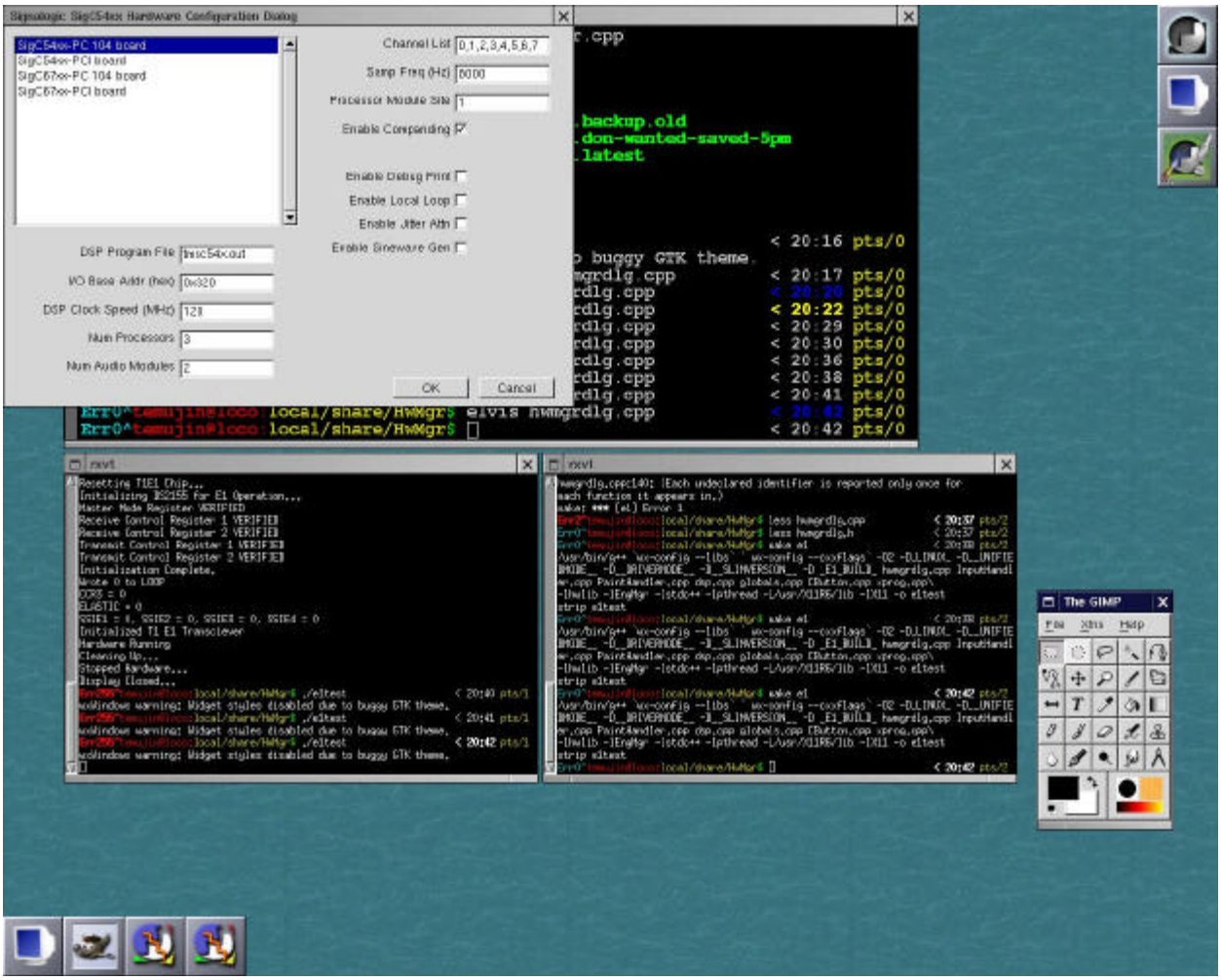


Figure 3-34, Hardware Manager Dialog, with Active Display in Background

3.5.3.4 Running the DUART Test Program

***NOTE**: The following instructions assume that the SigFM.o module has been built from the SigFModule directory. If not, see software build instructions in section 5.3, Building Linux Software.*

After initial system reboot, the kernel module must be loaded. See instructions in section 2.2.5 above, Linux Driver Install Instructions.

Before running the DUART test program, Hyperterminal must be set up on the PC to which the RS-232 adapter is connected.

To set up Hyperterminal, first run Start->Programs->Accessories->Communications->Hyperterminal.

Type in any name for the session and click OK.

Select the COM port the serial cable is attached to and click OK.

Select 19200 bit rate, 8 data bits, no parity, 1 stop bit, and Xon/Xoff Flow Control. Click OK.

To run the DUART test program, first enter:

```
shell/> /sbin/insmod SigFM
```

This will install the SigFM.o module into the kernel so that the Signalogic software can run.

Then move to the subdirectory from which the DUART test program will be run, as follows:

```
shell/> cd /usr/local/share/Comm-Prog
```

To run the Signalogic DUART test program, enter:

```
shell/Comm-Prog> ./CommProg
```

A prompt should appear asking whether to run the program in normal mode or loopback mode. Type 1 for normal mode to generate character output internally or 2 for loopback mode to generate character output from character input.

This will run the DUART test. If DUART runs successfully, the screen will appear as shown in Figure 3-35 below:

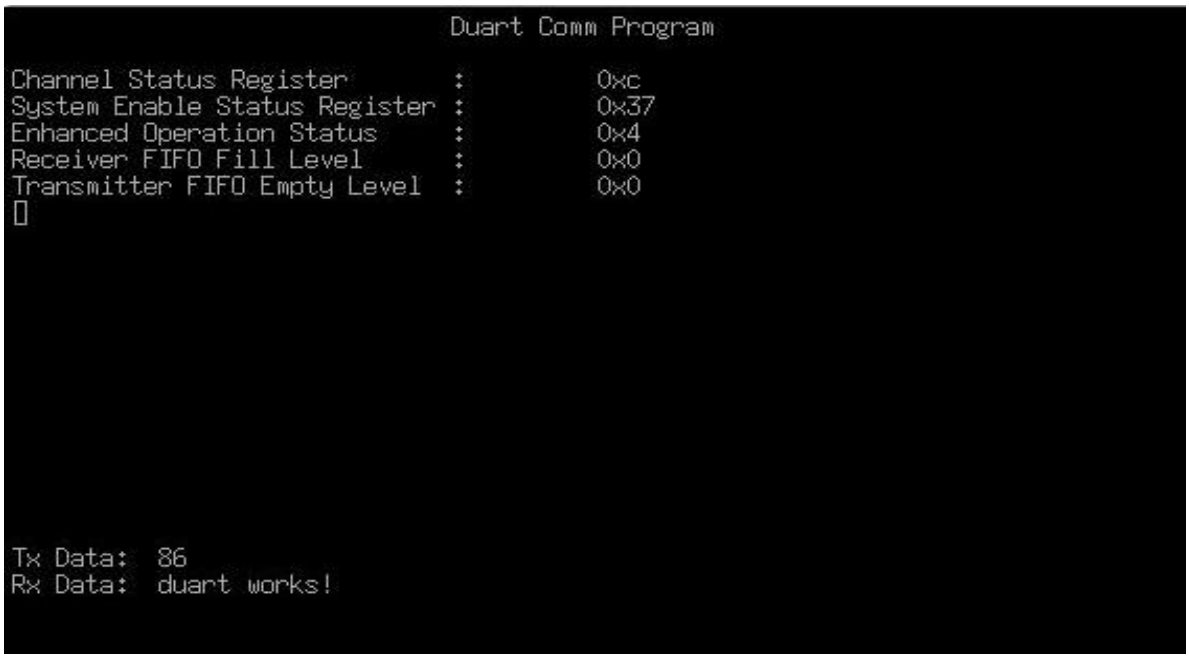

Duart Comm Program
Channel Status Register : 0xc
System Enable Status Register : 0x37
Enhanced Operation Status : 0x4
Receiver FIFO Fill Level : 0x0
Transmitter FIFO Empty Level : 0x0
Tx Data: 86
Rx Data: duart works!

Figure 3-35, Running DUART Comm Program

Note the values of the control or status registers should be 0xc, 0x37, 0x4, 0x0, and 0x0 respectively. For mode 1, *Tx Data:* should display numbers in a cycle from about 70 to 90. This means the DUART is transmitting capital letters from A to Z to Hyperterminal in a continuous cycle as shown below in Figure 3-36 below. When a key is pressed on the PC, the character should appear in Hyperterminal, and Rx Data should display the character pressed.

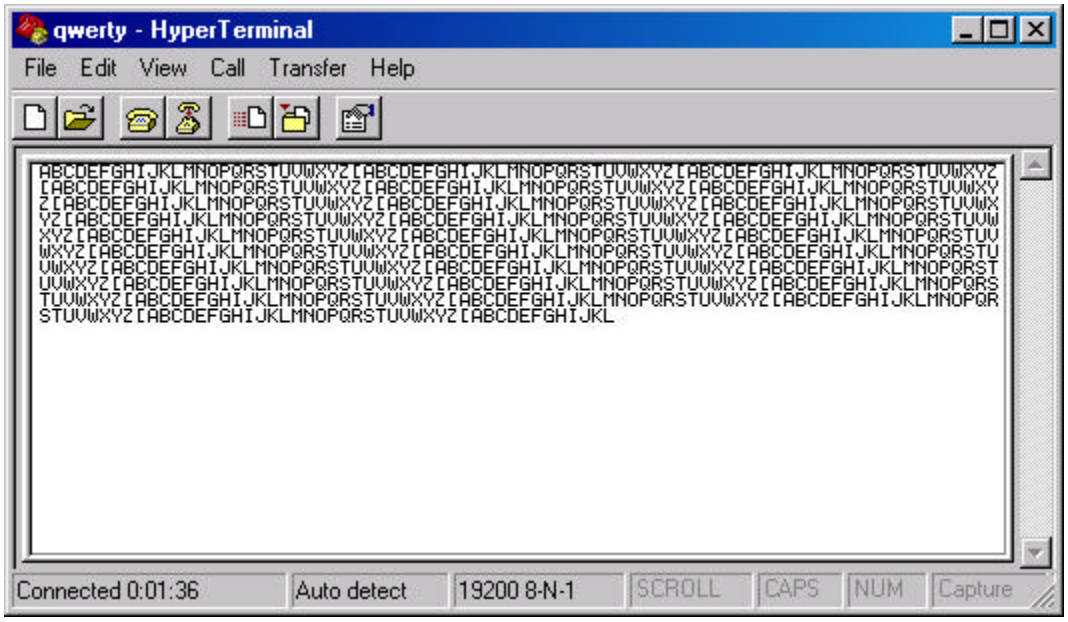


Figure 3-36, Hyperterminal on Target PC

3.5.3.5 Running PTMC Card Memory Test Programs

SigC641x and SigC5561 PTMC cards offer high-performance PCI-to-HPI data transfer. Memory test programs are provided to

- measure data transfer performance using various block sizes
- fully test onchip and offchip memory address ranges of each DSP device on the board, including “broadcast mode” for multiple DSPs and other memory test types

Two test program modes are available:

- Single-pass
- Diagnostic

In diagnostic mode, results for all specified DSP are displayed as continuously updating, which allows the test to run for an indefinite time period and show errors on cumulative basis.

Below is shown the command-line options for the memory test program, and example of a valid command-line entry:

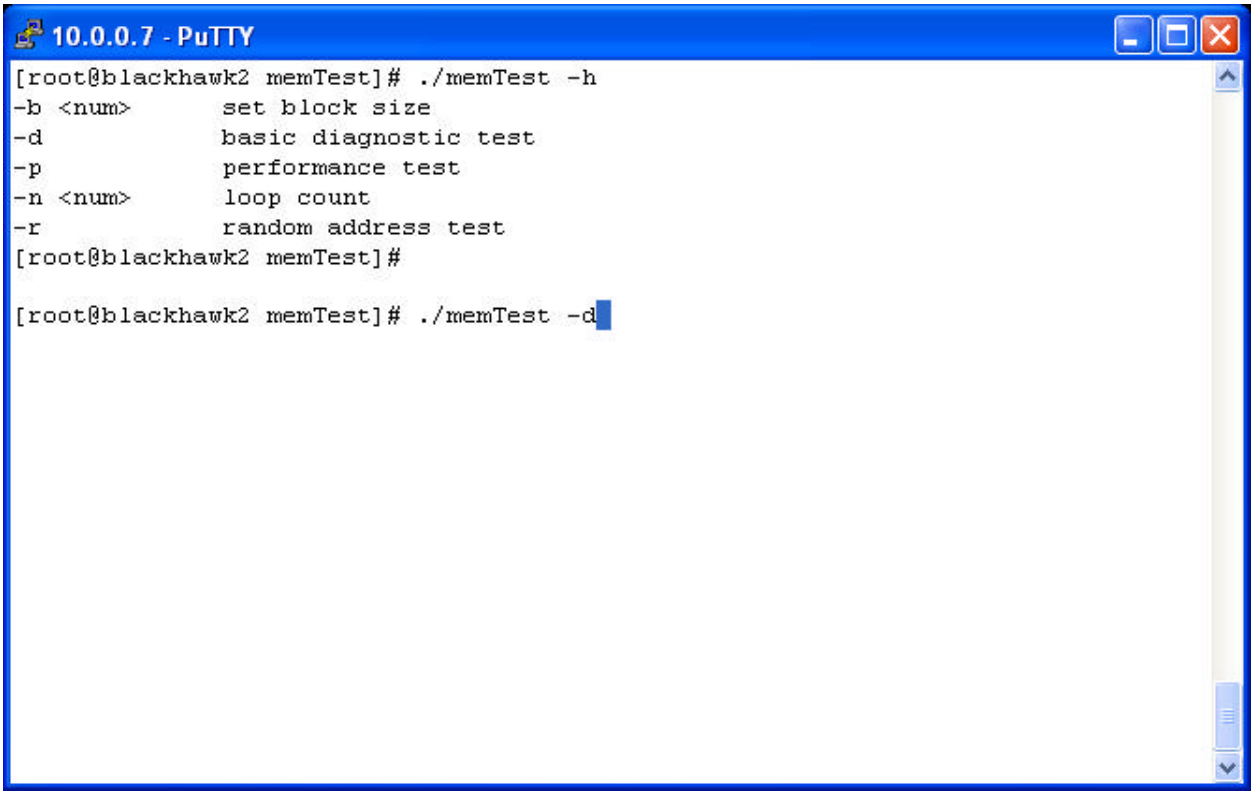


Figure 3-37, memTest Command-Line Usage and Example

 Copyright © Signalogic 2003-2005

Below is a screen capture of memTest output in random “block by block” mode. In this mode, the diagnostic writes one block of DSP memory, writes another randomly chosen block with different data, then reads back the first block. This test mode tends to emulate practical video and voice applications.

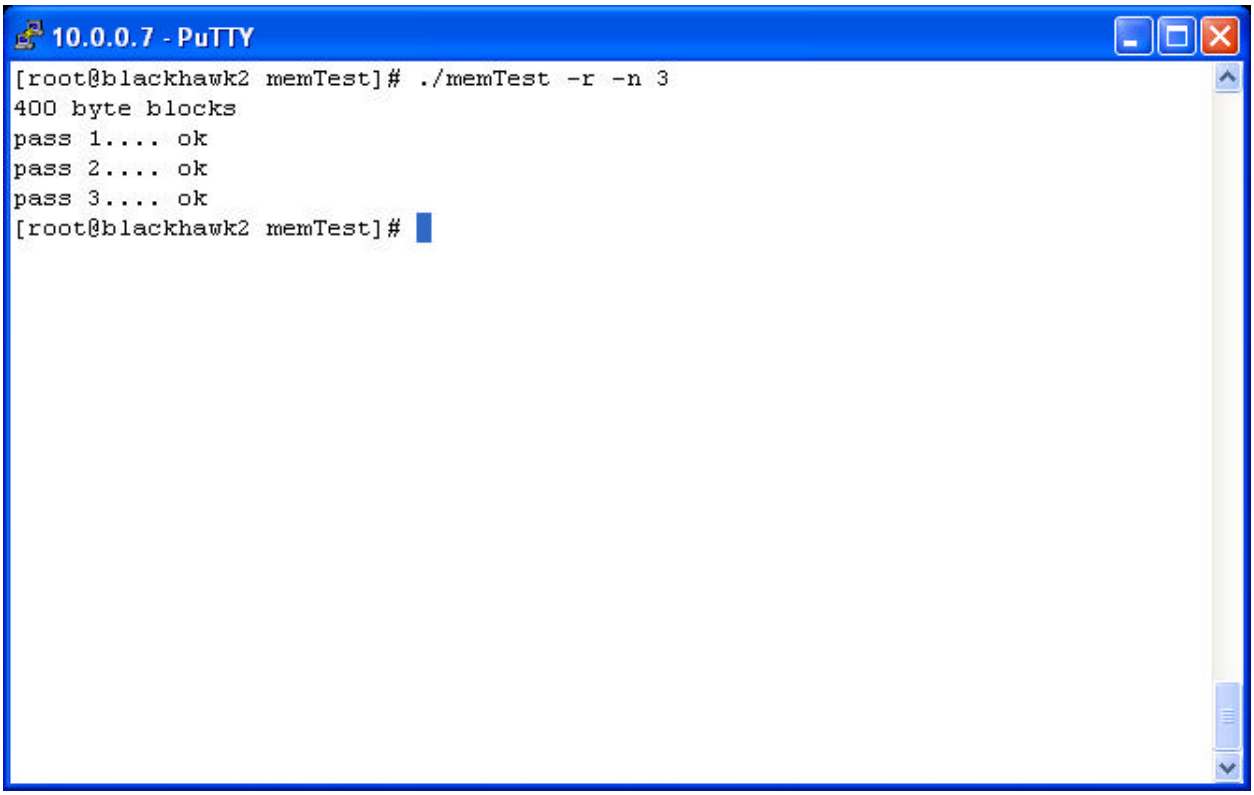


Figure 3-38, Running PTMC Card Memory Test Program, Random Block Test in Single-Pass Mode

Below is a screen capture of memTest with performance measurements shown:

 Copyright © Signalogic 2003-2005

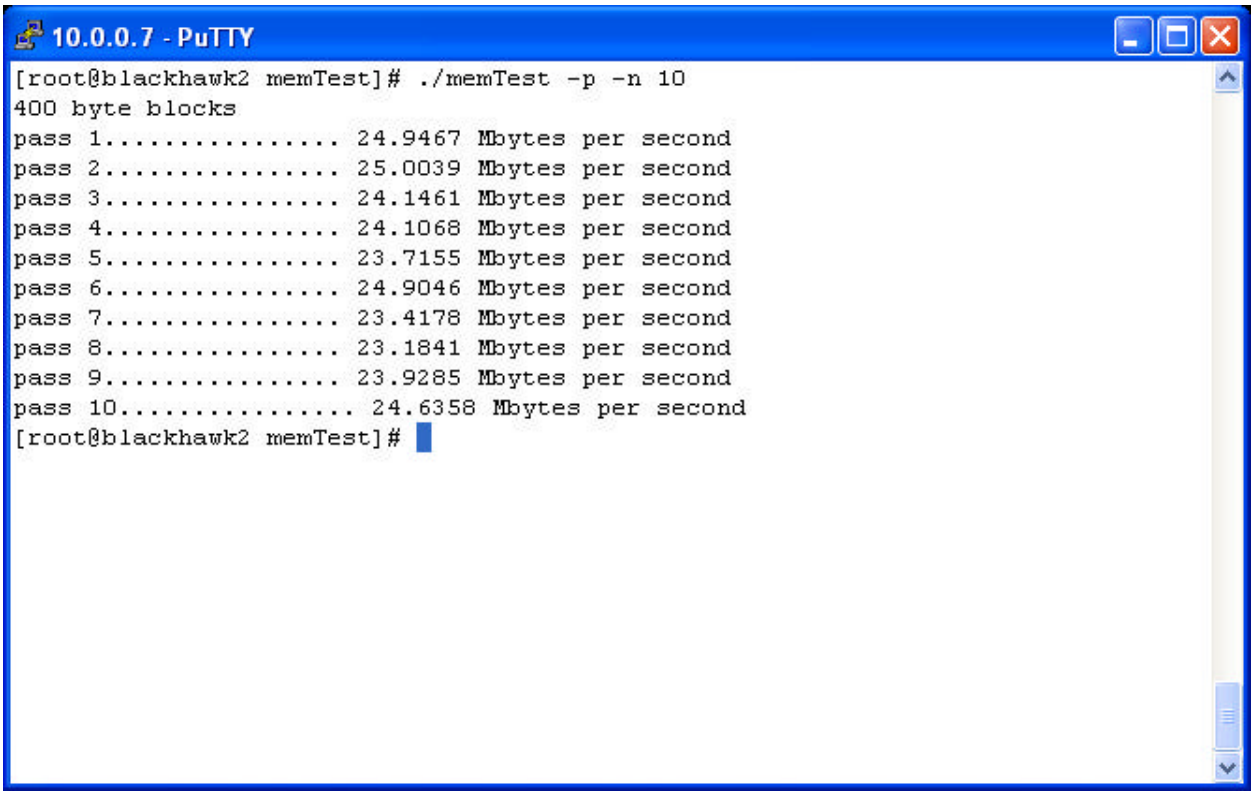

10.0.0.7 - PuTTY
[root@blackhawk2 memTest]# ./memTest -p -n 10
400 byte blocks
pass 1..... 24.9467 Mbytes per second
pass 2..... 25.0039 Mbytes per second
pass 3..... 24.1461 Mbytes per second
pass 4..... 24.1068 Mbytes per second
pass 5..... 23.7155 Mbytes per second
pass 6..... 24.9046 Mbytes per second
pass 7..... 23.4178 Mbytes per second
pass 8..... 23.1841 Mbytes per second
pass 9..... 23.9285 Mbytes per second
pass 10..... 24.6358 Mbytes per second
[root@blackhawk2 memTest]#

Figure 3-39, PTMC Card Memory Test Program Display, Performance Measurement in Single-Pass Mode

Below is a screen capture of memTest in Diagnostic mode. In diagnostic mode, the program continues to run the specified test and display results indefinitely, until an exit command is given.

 Copyright © Signalogic 2003-2005

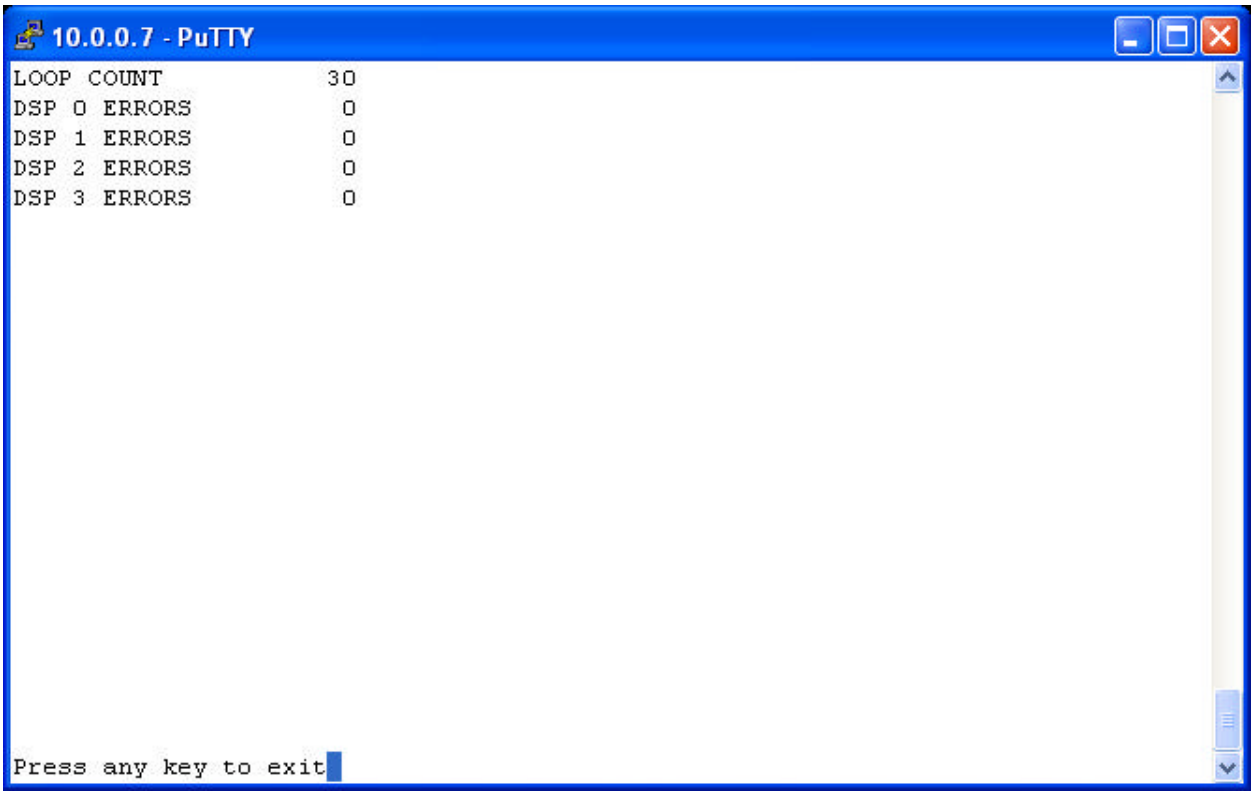


Figure 3-40, Running PTMC Card Memory Test Program, Diagnostic Mode

 Copyright © Signalogic 2003-2005

4 Building DSP Application Software

Building DSP application software requires Code Composer Studio (CCS) software from Texas Instruments. Here are some important notes about CCS software and the example CCS projects that Signalogic provides:

- For every hardware type and DSP processor type, there are two (2) default projects:

  SCI C5xxx or C6xxx SCI-only code build
  DSP_BIOS SCI combined with DSP/BIOS code build

- Low-level boot-code detects stand-alone mode vs. host mode automatically and adjusts boot configuration and code initialization as needed

The SCI builds are the most memory-frugal and highest performance options. The DSP_BIOS builds offer full compatibility with Texas Instruments “DSP/BIOS” kernel and RTOS, including SEM (semaphore) and TSK (task) objects, CSL (Chip Support Library), and other TI support tools.

More information on host mode vs. stand-alone mode operation is contained below in section 4.2, Host Mode vs. Stand-Alone Mode.

4.1 CCS Project Build & Run Sequence

This section assumes you have already installed Code Composer Studio (CCS) v2.0 or higher for C5000 and/or C6000 series. If not, then see section 2.1 above, Installing Code Composer Studio Software.

Below are step-by-step instructions to start CCS, open a project, make changes, build the project, download the new binary output file (containing executable code) to the hardware, and run the new code.

1) Starting CCS. To start CCS, click on CCS 2 icon on the desktop or click Start button at lower left of your desktop and go through this men sequence:

Programs | Texas Instruments | Code Composer Studio

Note: Possible JTAG Emulation Error Message

If the following JTAG emulations error message box (Figure 4-1 below) appears when running CCS, click the Ignore button and proceed. CCS will still allow you to edit, compile, and link (build) a project without the JTAG emulator working correctly. A temporary JTAG emulator error condition can happen frequently, for example due to DSP board being turned off, incorrect configuration of the PC parallel port, or even just a "bad session" the previous time CCS was run, for example code crash due to NULL pointer or other routine error.

CPU_1 - DSP Device Driver
Can't Initialize Target CPU
I/O Port = 240
Board Name: Heterogeneous Multi-Target (Texas Instruments)
Cpu Name: CPU_1
Abort: Close Code Composer Studio.
Retry: Try to initialize the target again.
Ignore: Ignore the initialization error and start without target debugging access.
Abort
Retry
Ignore

Figure 4-1, Possible JTAG or Hardware Error Message when Opening CCS

2) Opening a Project. Select Project | Open on main CCS menu, as shown in Figure 4-2 below.

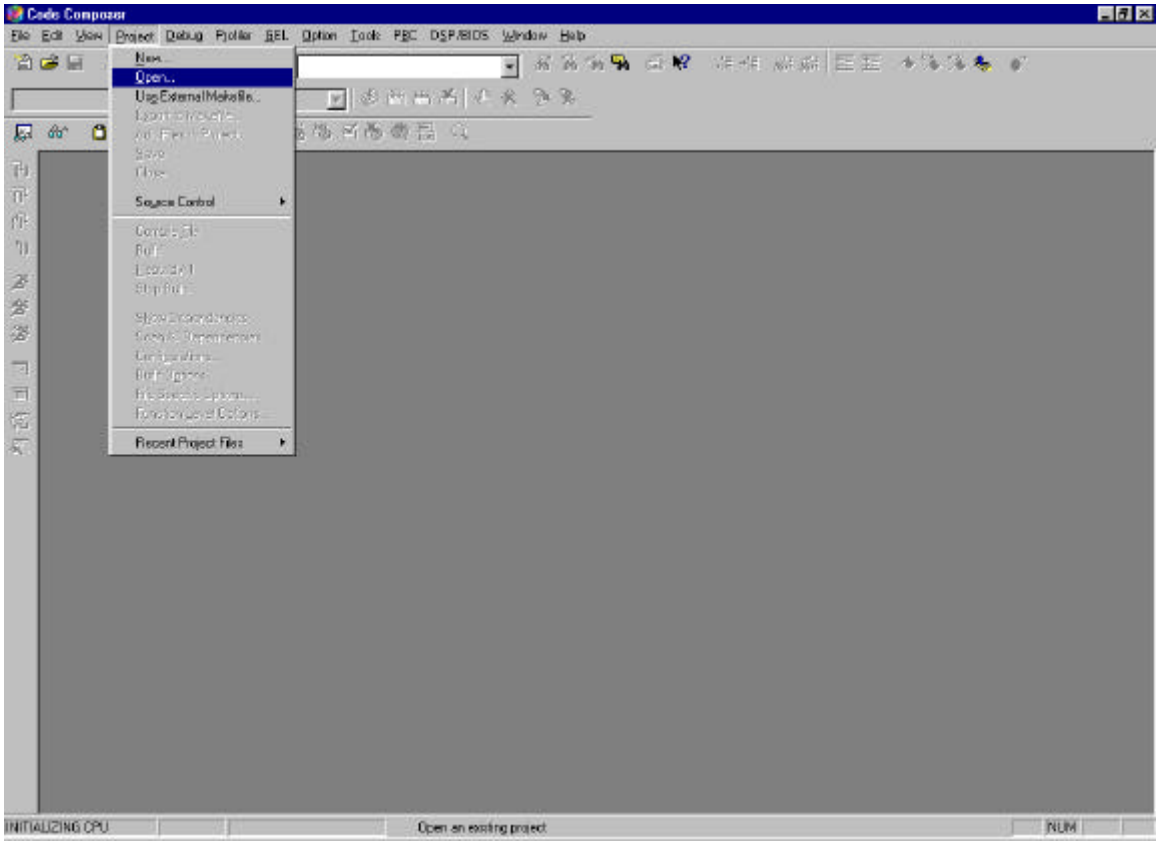


Figure 4-2, Opening a Project in CCS

3) Selecting the Correct Project File. To select the correct project file, depending on hardware and build type, use the table below as a guide:

Hardware	**Build**	**Path**	**Project File (.pjt)** [1]
SigC54xx PC/104 SigC54xx-PCI DSK C5402	C54xx SCI	\dspcode\c54xx\SCI	C54xx_SCI
	C54xx DSP/BIOS	\dspcode\c54xx\DSP_BIOS	C54xx_SCI_DSPBIOS
SigC6xxx-PC/104 SigC6xxx PCI DSK C6x11	C6xxx SCI	\dspcode\c6xxx\SCI\	C6x11_SCI
	C6xxx DSP/BIOS	\dspcode\c6xxx\DSP_BIOS\	C6x11_SCI_DSPBIOS
SigC64xx-PTMC	C64xx SCI	\dspcode\c6xxx\SCI	C64xx_SCI
	C64xx DSP/BIOS	\dspcode\c6xxx\DSP_BIOS	C64xx_SCI_DSPBIOS

M6x [2]	C6xxx SCI	\dspcode\c6xxx\SCI	C6x01_SCI
	C6xxx DSP/BIOS	\dspcode\c6xxx\DSP_BIOS	C6x01_SCI_DSPBIOS

Notes About Project Files

[1] If you are running a version of Code Composer Studio *lower than v2.1*, the extension for project files is **.mak**, not **.pjt**

[2] A separate project file is provided for M6x hardware as several additional files are included in the M6x board projects that provide support for the numerous I/O modules supported by M6x boards. I/O module examples include A4D4, SD4, SD16, AIX, etc.

Some examples of opening projects are given in the figures below. The examples show C6xxx versions being opened, but the same procedure applies to C54xx. Note that the Project Open dialogs in the figures have been edited (see red highlighted areas) to show the full "dspcode" subfolder hierarchy, although the normal Win2K or WinXP style for Open dialogs does not include that.

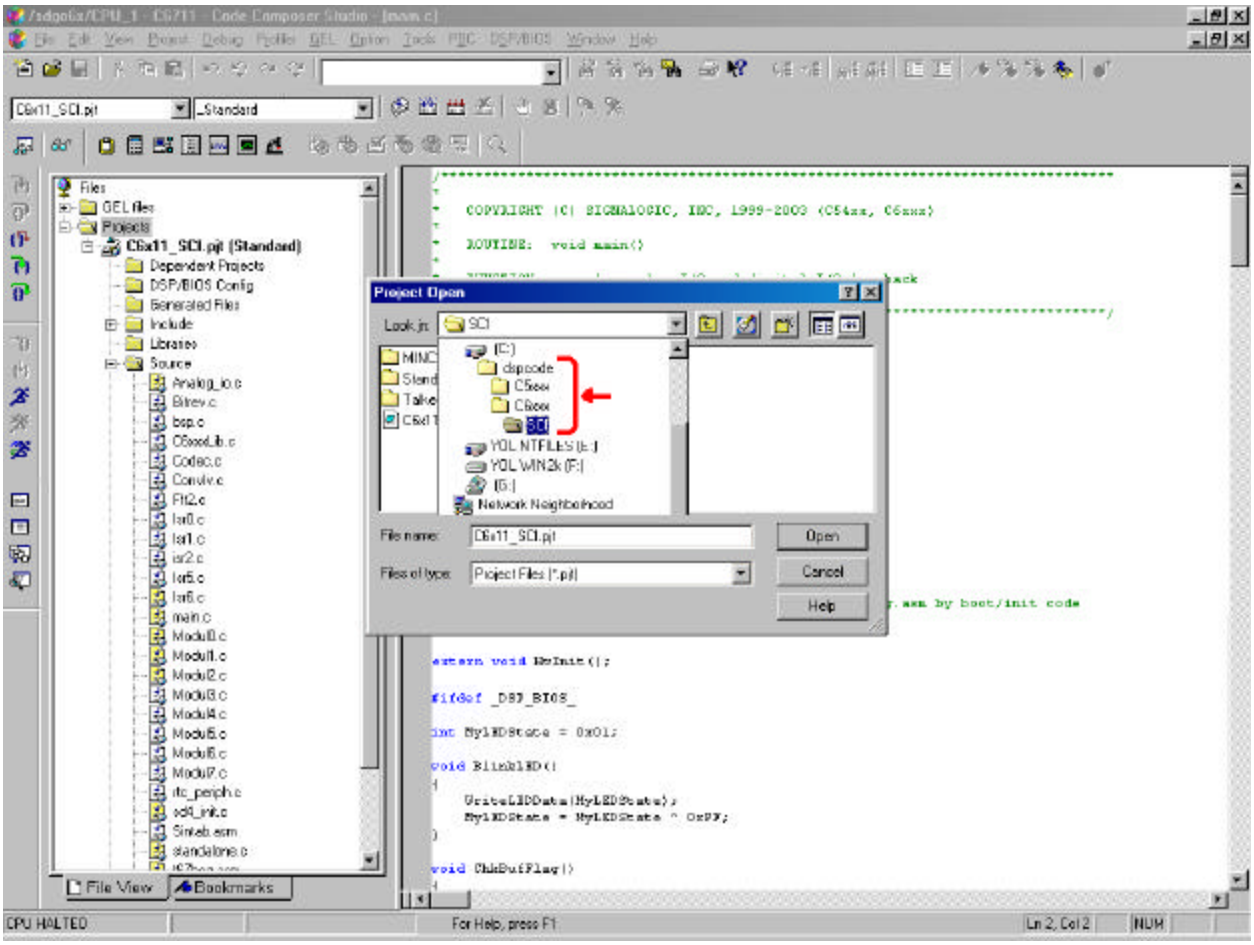


Figure 4-3, Opening the SCI Project

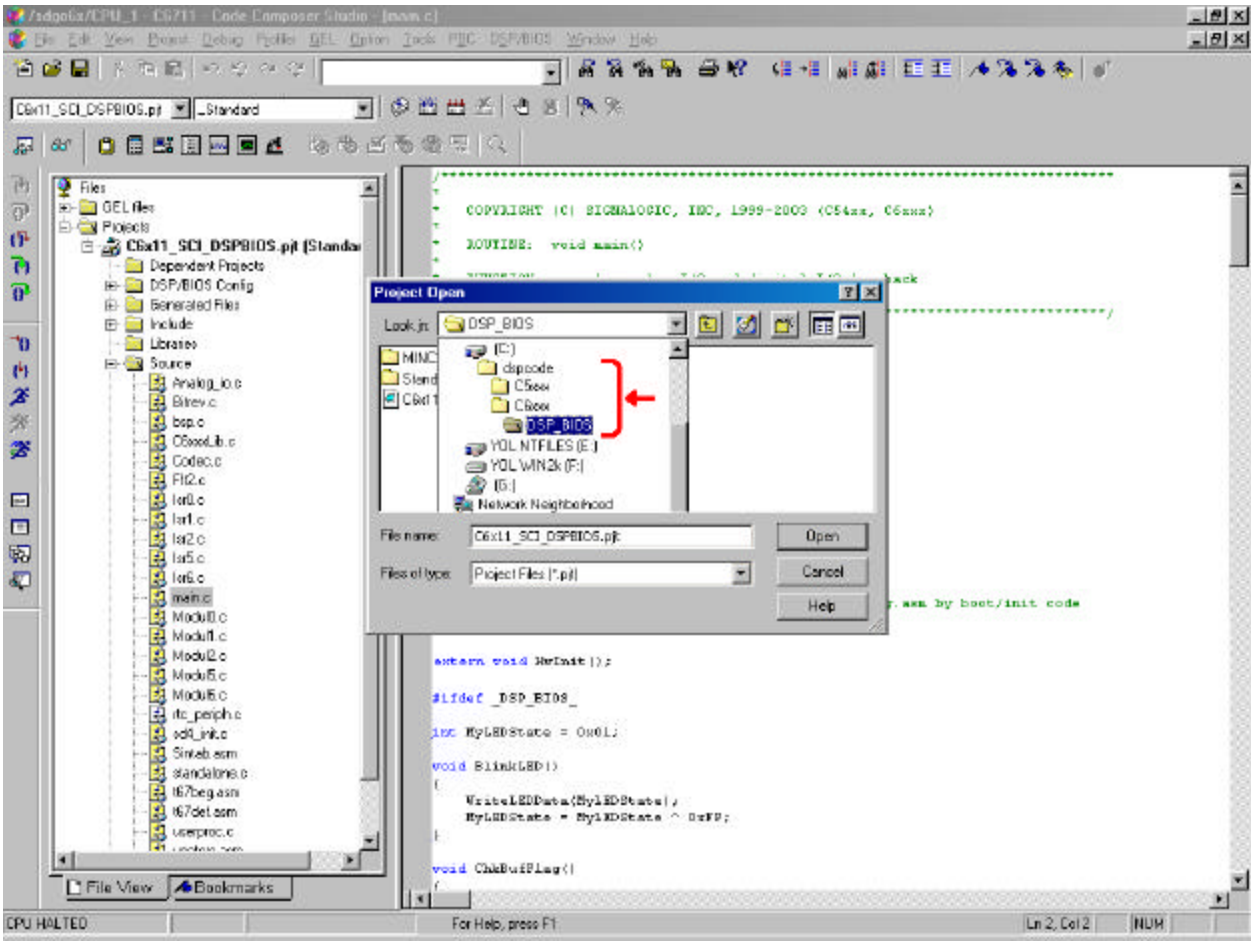


Figure 4-4, Opening the DSP_BIOS Project

After loading the desired project file (.pjt file), the Project View window at left of the IDE will display the name of the project and all files associated with the project, including C and assembly language source, include files (.h files), and auto-generated files such as .cdb or other files generated by the CSL (Chip Support Library) or DSP/BIOS.

Notes About CCS IDE

1) The standard Windows tree display is used in the Project View window; clicking '+' will display the contents of a project or directory, clicking '-' will collapse the contents into the project or directory.

2) Include files may not be manually added or deleted from the File View tab of the Project View window. Include files are determined by CCS automatically, as a result of automatic dependency check performed at build time. If you want to delete a .h file, be sure it is not referenced in any .c or other source file.

4) Editing Source Code Files. To edit C (or assembly language) files in the project, double-click on a file listed under Source or Include categories in the project view at left of the CCS

main window. This will display the source code file in the File Display window, where it can be edited as needed. Examples of editing both C and assembly language source code files are shown in Figures 4-5 through 4-7 below.

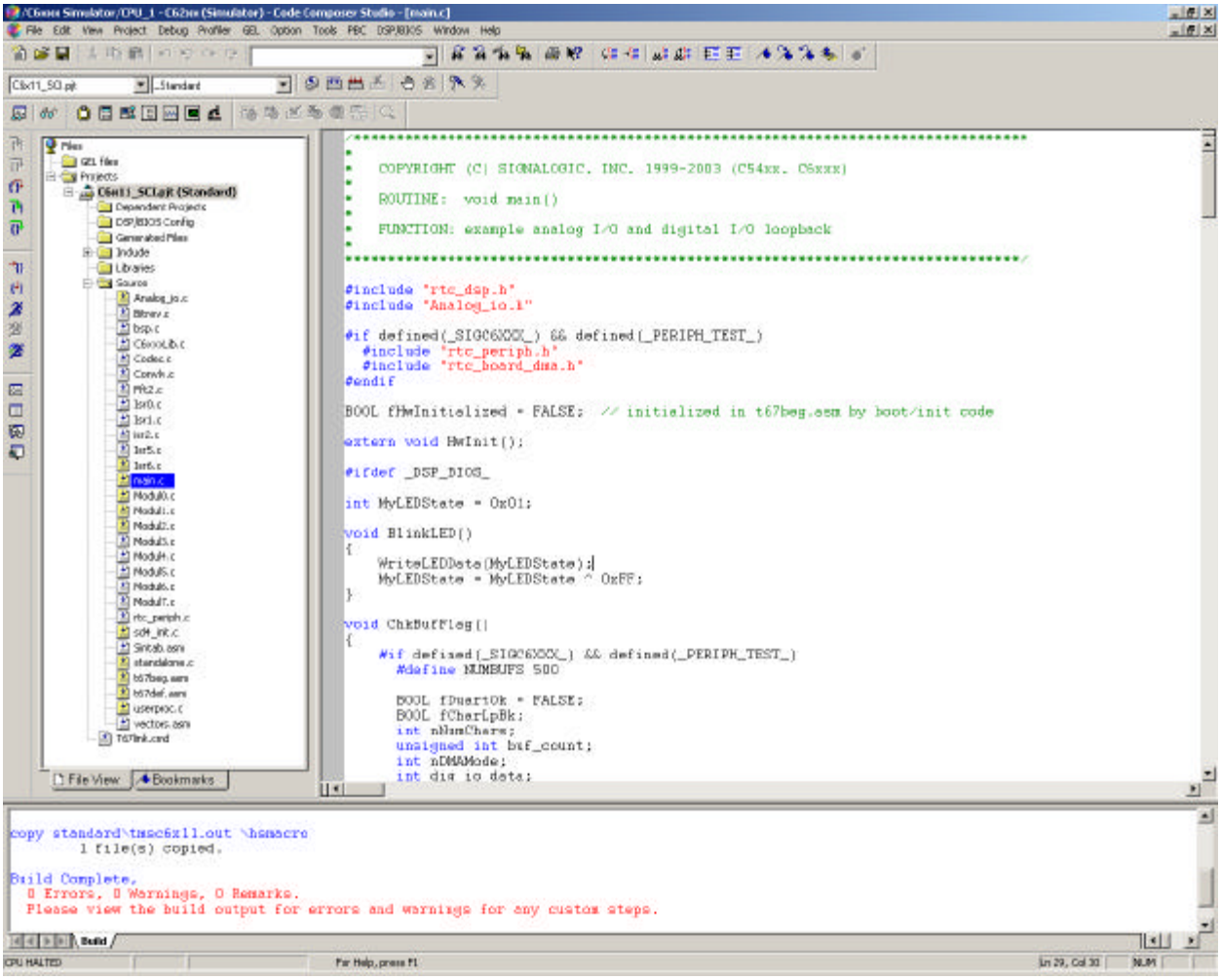


Figure 4-5, SCI project opened, with main.c source code highlighted

Below is a similar example, with the DSP_BIOS project opened instead. Note again that in many of these examples, C6xxx projects are shown, but the subfolder hierarchy and project naming convention is identical for C54xx projects.

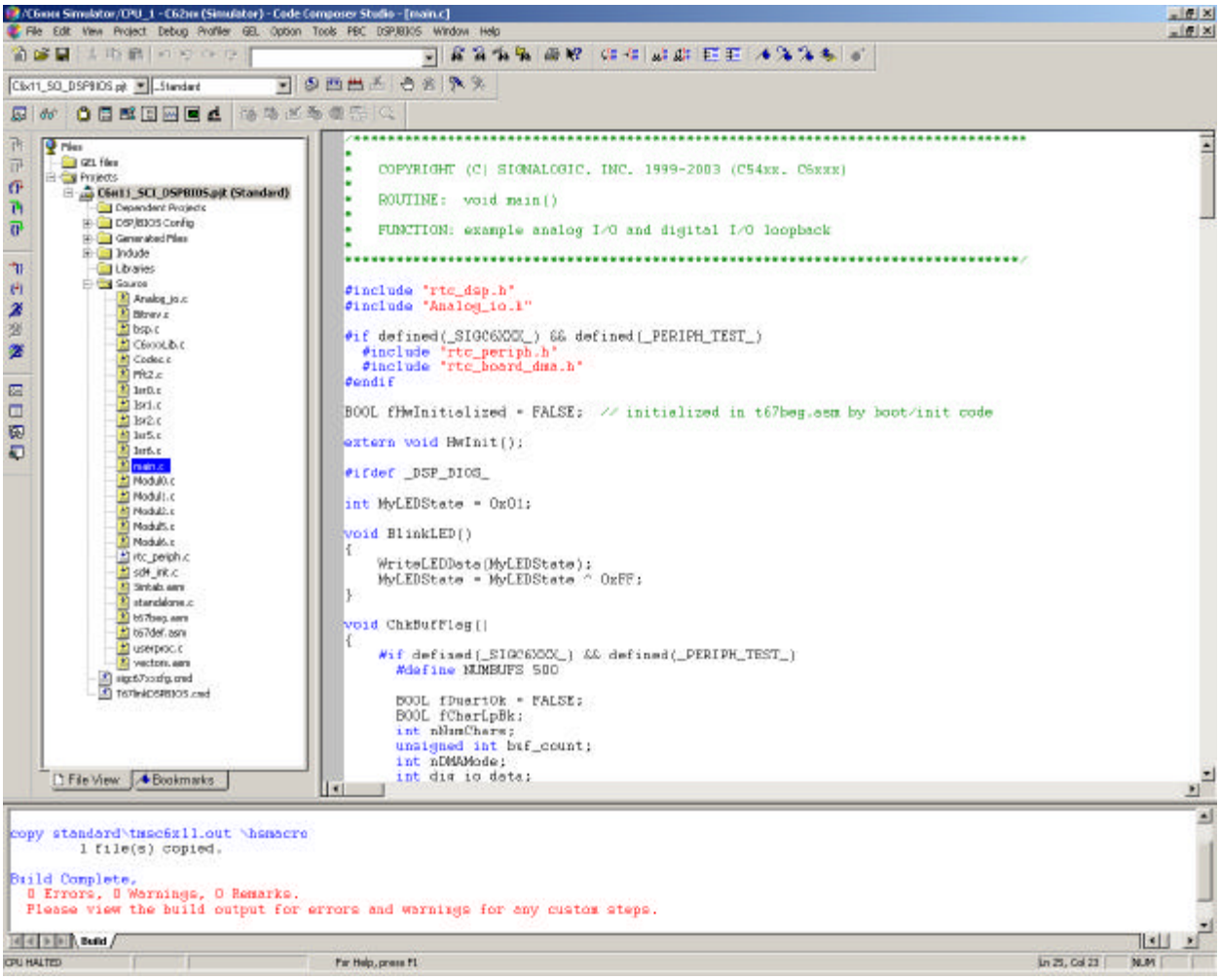


Figure 4-6, DSP_BIOS project opened, with main.c source code highlighted

Below is an example of an assembly language source code file open and displayed. Assembly language source may be edited and debugged in CCS without limitation. For example when single-stepping using the CCS JTAG-based debugger, the Program Counter (PC register in CCS Core Register View window) will advance to individual instructions rather than lines of code as in C source. Breakpoints and other debug features are likewise available when debugging at the assembly language level.

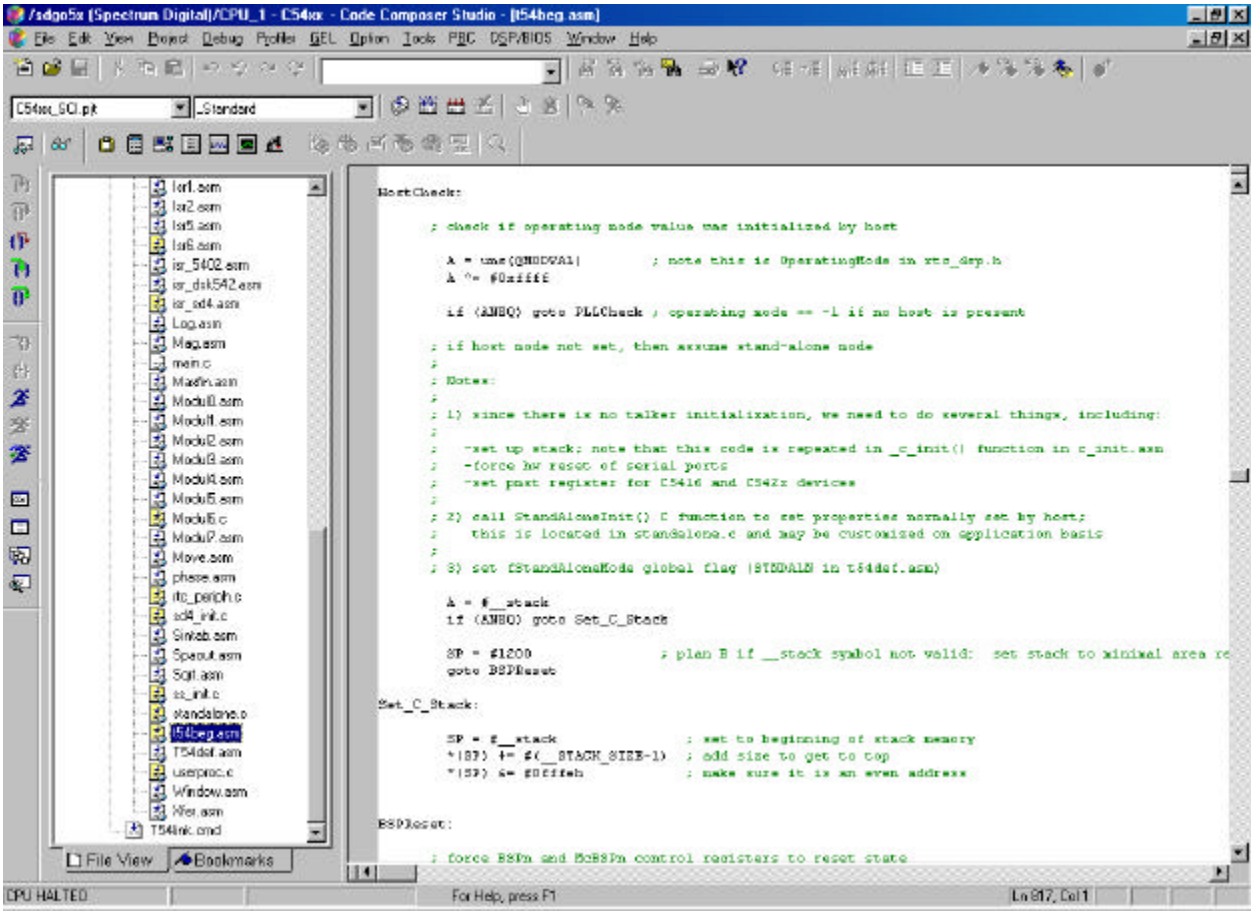


Figure 4-7, Assembly Language Source File Opened

5) Changing Build Options. To change project build options, select Project | Build Options on the CCS main menu. A dialog box titled “Build Options” should appear, as shown in Figure 4-8 below.

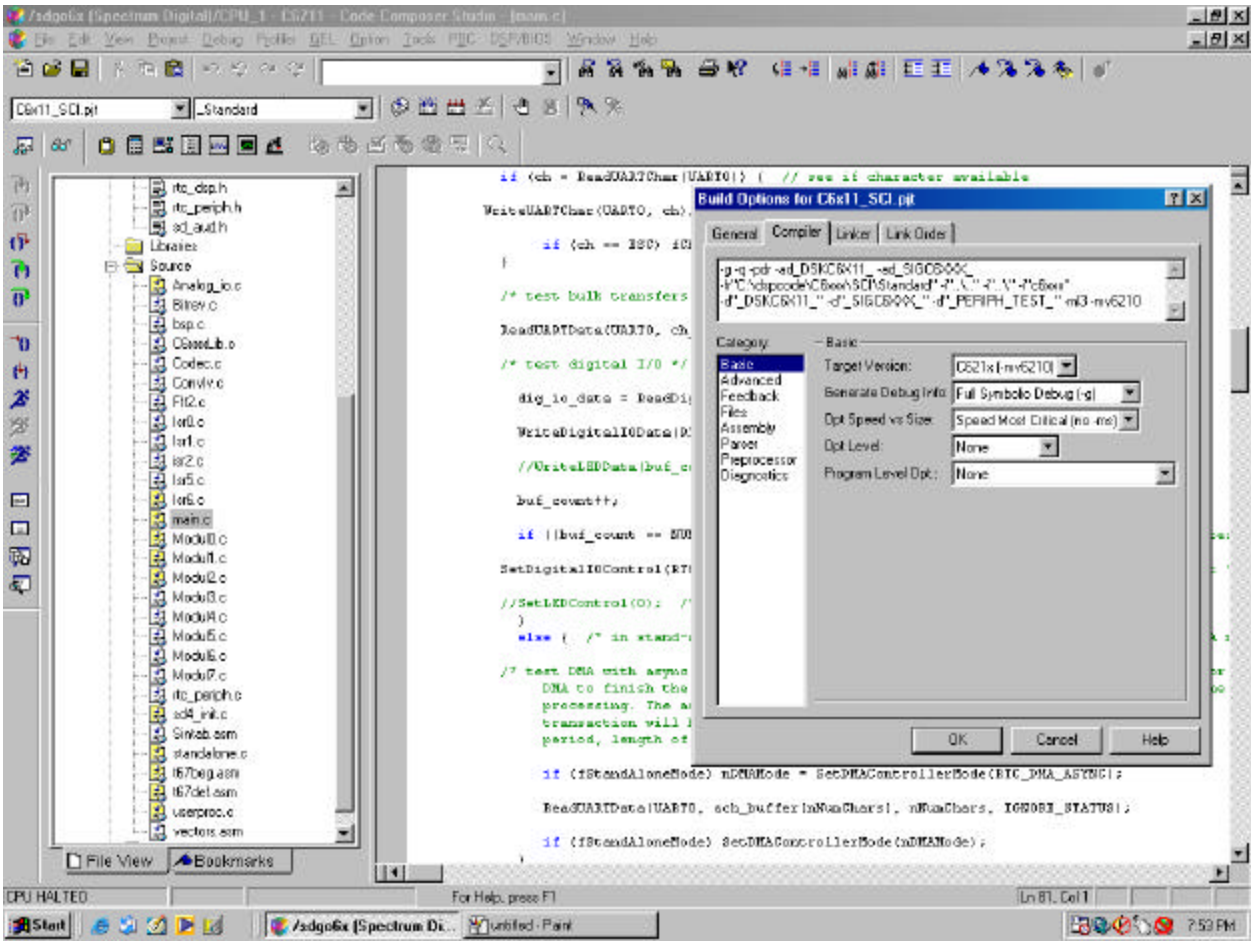


Figure 4-8, Project Build Options Dialog

6) Changing DSP/BIOS Options. To view or change DSP/BIOS options, double-click on the ".cdb" file shown under the "DSP/BIOS Config" subfolder in the File View window at left. ***You should only see a .cdb file if you have one of the available xxx_SCI_DSPBIOS project files loaded!*** (In the project naming convention, xxx refers to hardware type; see the table in step 3 above, Selecting the Correct Project File).

After double-clicking, you should see a display similar to Figure 4-9 below.

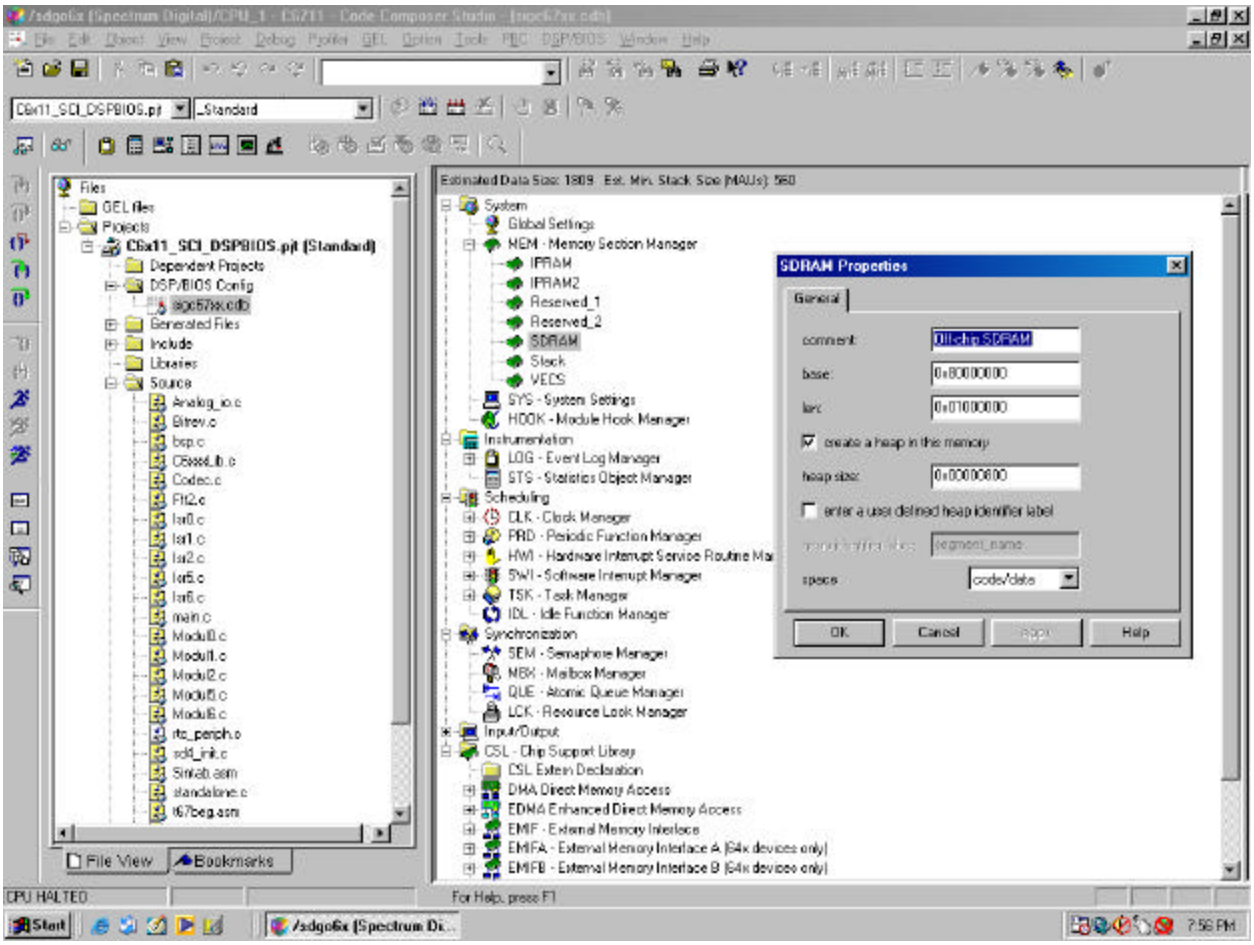


Figure 4-9, Project Build Status Display

In the DSP/BIOS configuration display example above, Properties for the SDRAM entry under the "MEM - Memory Section Manager" item are displayed. Note that the SDRAM definitions conform to the physical memory on SigC67xx multi-DSP modules.

7) Building a Project. To build the project, select Project | Build or click on the build icon on the toolbar. If the build is successful, you will see status messages at the lower window in the CCS IDE, similar to those shown in Figure 4-10 below.

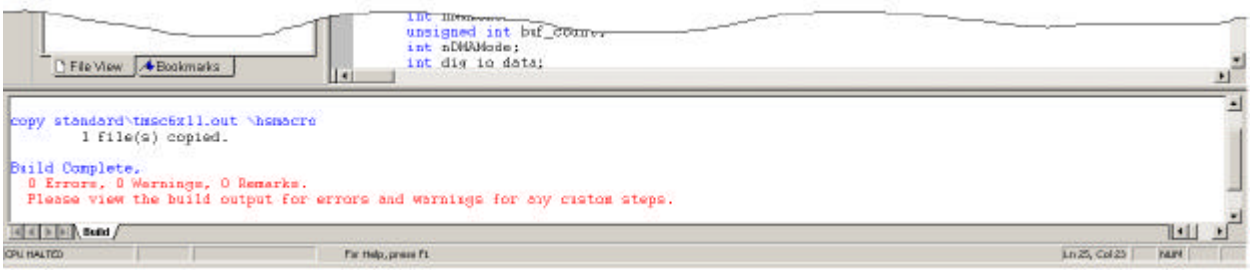


Figure 4-10, Project Build Status Display

 Copyright © Signalogic 2003-2005

Building the project will create a DSP executable (binary) file in COFF format. After the build is complete as shown in Figure 4-10 above, the DSP executable file can be downloaded and run in:

- Stand-Alone Mode, using CCS and JTAG emulator hardware

…or…

- Host Mode, using Hypersignal software or a Visual C/C++, Visual Basic, or MATLAB application using DirectDSP software to interface to the DSP hardware

Notes

1) As one of the last build steps, the resulting DSP executable file is copied to the \hsmacro subdirectory to support Host Mode operation (this step is visible in Figure 4-10 above).

2) Low-level boot code (t67beg.asm) auto-detects host and stand-alone mode at run-time.

For more info about…	**See section…**
Host Mode vs. Stand-Alone Mode	4.2
Talker (host interface driver)	4.1.4
DSP executable files and locations	4.1.7
COFF file format	4.1.8

 Copyright © Signalogic 2003-2005

4.1.1 SCI Builds

SCI (Source Code Interface) builds use only Signalogic-provided framework files and contain no operating system kernel or other DSP/BIOS objects. The SCI builds provide higher performance, smaller memory footprint; and less protected and restricted access to DSP chip and board resources.

SCI framework files include:

- Host interface driver
- Board peripheral I/O drivers and other board resource files
- DSP and math functions
- Real-time I/O and C code signal processing examples

For SCI project file names, see step 3 in section 4.1 above, labeled "Selecting the Correct Project File", for more information.

4.1.2 DSP/BIOS Builds

DSP/BIOS builds combine Signalogic-provided SCI (Source Code Interface) framework files with the Texas Instruments "DSP/BIOS" kernel and real-time operating system. DSP/BIOS builds offer somewhat less performance and require substantially greater memory footprint; but are more suitable for larger projects involving a high amount of application C code and possibly several tasks and intertask communication.

C6xxx DSP/BIOS Notes

Currently the C6xxx DSP/BIOS build fits all heap and stack objects into onchip memory; i.e. less than 64kbyte for C6x11 processors. However, that leaves little onchip memory for application memory, requiring offchip SDRAM to be used for application data and buffers.

While onchip SRAM offers zero wait-state performance in most cases, SDRAM forces a penalty of several wait-states on first access and 1 to 2 wait-states for subsequent sequential accesses.

SDRAM clock speed on the SigC67xx module is 75 MHz (processor clock / 2).

If memory speed becomes a critical issue and SDRAM performance leaves the application short of real-time operation, then it may become necessary to use the non-DSP/BIOS build and relocate application data and buffers into onchip SRAM.

4.1.3 MINCONFIGx Active Configurations

The MINCONFIGx Active Configurations are available in both SCI and DSP/BIOS projects. They are intended to produce small-memory-footprint output, trading off reduced functionality. Below is a table showing this tradeoff:

Active Configuration	Operating Modes	Host Interface ("Talker") Enabled	DSP and Math Functions	Mode 6 Functionality
cStandard	All	Yes	All	Full
MINCONFIG	0, 1, 2, 6	Yes	Reduced	Reduced
MINCONFIG2	6	No	None	Reduced

The Active Configuration field in the CCS is at the upper left of the IDE, as shown in Figure 4-11 below (see red highlighted area). For all projects, the default setting is "cStandard", but can be set to "MINCONFIG" or "MINCONFIG2", as indicated in the above table.

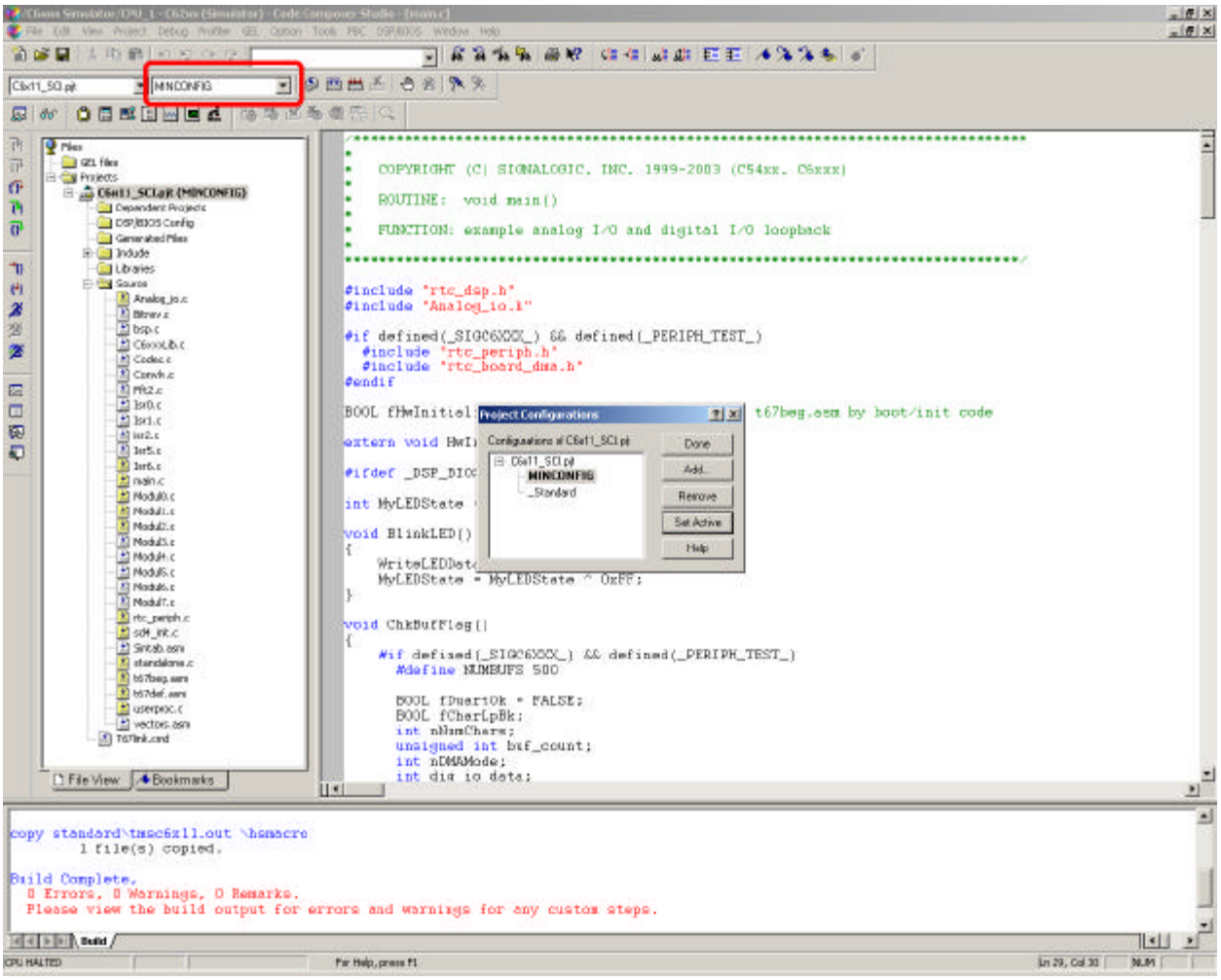


Figure 4-11, MINCONFIG Active Configuration Selected

4.1.4 Talker – Host Interface Driver

A small DSP executable known as "Talker" serves as a host interface driver for both SCI and DSP/BIOS builds. Talker is, by definition, a tiny, highly optimized (hand-written assembly language) program that has a small impact on performance and memory usage of real-time DSP code.

The purpose of Talker is to allow host processor software to make efficient data transfers to/from DSP memory and peripherals. By using a small DSP program to control transfers, the host is able to "see" all areas of the DSP internal architecture, even those that might be restricted from direct HPI access.

The choice of whether to use talker or not is made by the host software application layer. If the DSInitBoard() or DSInitProcessor() DirectDSP API calls are made, then talker will be loaded to DSP memory and initialized.

All SCI and DSP/BIOS project linker command files reserve a small "memory" hole for a talker program, in case it should be used.

Below is a list of talker programs and source code, by board type.

All SCI software includes talker source and a small CCS project file for rebuilding Talker. However, Signalogic will provide no technical support for talker modifications and rebuild – this source code is provided strictly on "as-is" basis, with no additional information or engineering support.

4.1.5 Cache – Enabling and Organizing Code

Enabling cache in a DSP code build can increase signal processing performance greatly. The objective of cache is to allow a large amount of code to reside in slower SDRAM external memory, but execute fast when it is brought into onchip memory to run. Normally, external memory is slow by comparison to internal (onchip) memory; for example, SDRAM can provide a large amount of relatively low power consumption memory, but at a cost of slow performance, especially when it contains executable code. Cache provides the bridge mechanism to take advantage of external memory but still allow high performance when needed.

To enable cache for C6711 boards, the following must be set:

- DSP/BIOS build; use C6x11_SCI_DSPBIOS_CACHE project
- For C6711, MINCONFIG2 Active Configuration must be enabled; this allows onchip L2 memory to be configured as 1/2 SRAM and 1/2 cache
- Set _CACHE_ENABLE_ define in Project | Build Options (Preprocessor definitions). This is set by default in the C6x11_SCI_DSPBIOS_CACHE project

To enable cache for C6415 boards, the following must be set:

- DSP/BIOS build; use C64xx_SCI_DSPBIOS_CACHE project
- For C6415, MINCONFIG2 Active Configuration must be enabled; this allows onchip L2 memory to be configured as 1/4 SRAM and 3/4 cache
- Set _CACHE_ENABLE_ define in Project | Build Options (Preprocessor definitions). This is set by default in the C64xx_SCI_DSPBIOS_CACHE project

For an example of opening the C6x11_SCI_DSPBIOS_CACHE project, see Figure 4-12 below:

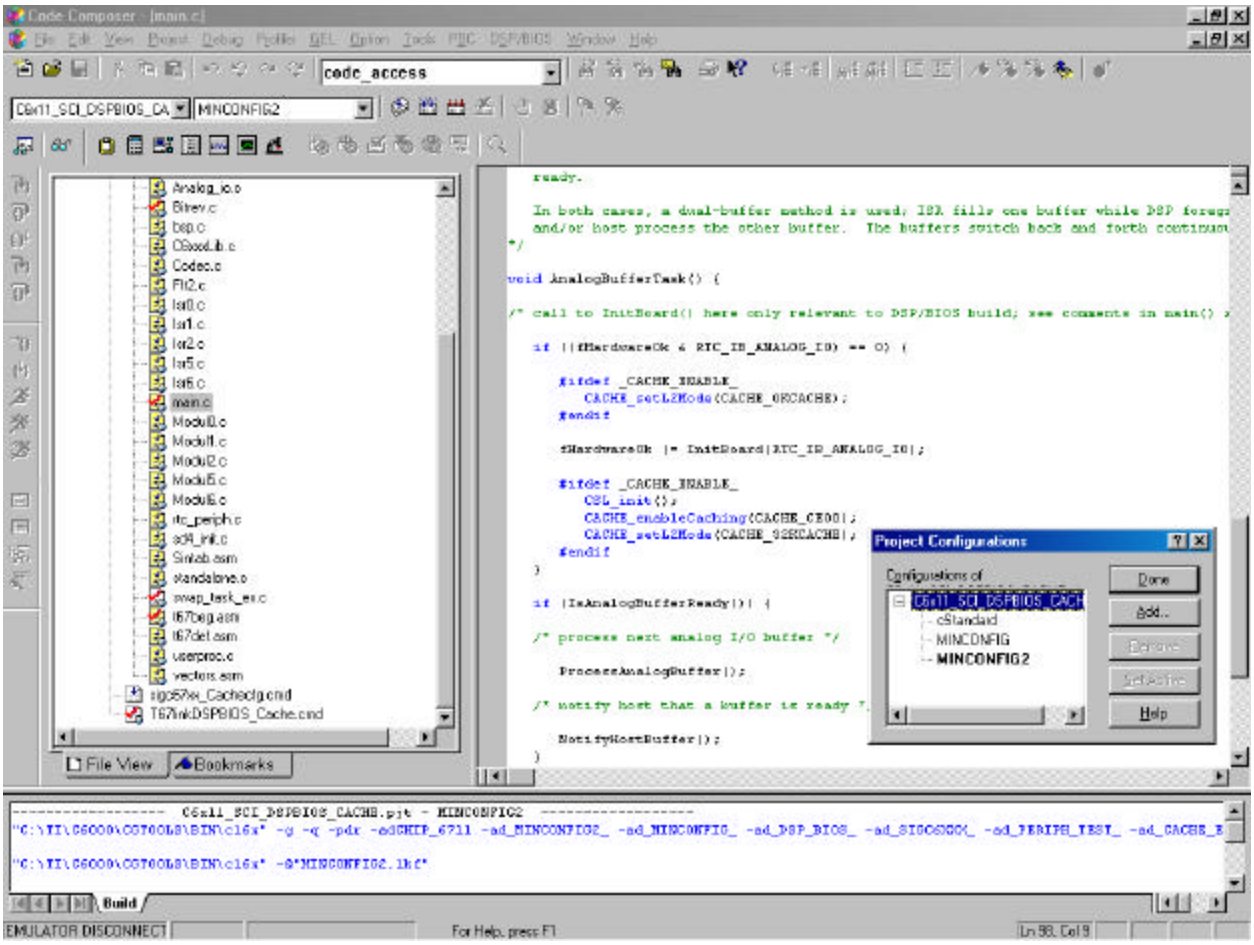


Figure 4-12, MINCONFIG Active Configuration Selected

Note that when the _CACHE_ENABLE_ define is set, additional code is activated in AnalogBufferTask(), as shown below in main.c:

```
#ifdef _CACHE_ENABLE_

#include <csl.h>
#include <csl_cache.h>
#include <csl_emif.h>

#endif

.
.
```

```
.

void AnalogBufferTask() {

.
.
.

   if ((fHardwareOk & RTC_IB_ANALOG_IO) == 0) {

#ifdef _CACHE_ENABLE_

      CACHE_setL2Mode(CACHE_0KCACHE);  /* reset CCFG, “zero K cache”, L2 mem all SRAM */

#endif

      fHardwareOk |= InitBoard(RTC_IB_ANALOG_IO);

   }

/* init and flush cache */

#ifdef _CACHE_ENABLE_

   CSL_init();
   CACHE_setL2Mode(CACHE_32KCACHE);
   CACHE_enableCaching(CACHE_CE00);
   CACHE_flush(CACHE_L2ALL,(void*)0x00000000, 0x00000000);

#endif
```

Some additional notes about the code above:

1) As indicated above, ***it is crucial to reset and disable cache prior to hardware initialization***. This is because some device drivers contain hardware-dependent code that is sensitive to timing and sequence, which can be altered and disrupted if cache is enabled.

2) The default settings in the C6x11_SCI_DSPBIOS_CACHE project are as follows:

- onchip L2 memory partitioned as 32k SRAM, 32k cache (applicable for C6711), or 64k SRAM, 192k cache (applicable for C6713)
- cache enabled for CE0 memory space (SDRAM memory area on SigC6xxx hardware)
- ***flush all cache prior to using it!*** Very, very important – this can fix some errata and bugs having to do with cache on C67xx devices

3) CSL indicates “Chip Support Library”. DSP/BIOS must be incorporated into the project in order to use the CSL.

Organizing Code to Take Advantage of Cache (and Avoid Cache Misses)

To fully take advantage of cache, you must organize signal processing code in a thoughtful way.

First and foremost, you have to define coherent sections of code that will reside in cache at one time, and not call other functions or branch to other sections of code that are not in cache. If not, then you will incur frequent cache misses, causing code to execute from slow external memory, and defeating the purpose of cache.

A classic example, is to try to organize nested loops so they will run entirely in cache – and if the loop code makes function calls, then to minimize the number of function calls (or combine into fewer functions), and ensure the function code can also fit into cache at the same time.

For example, inside UserProc(), code might be organized as follows:

```
UserProc(int* pIn, int* pOut, int nLen, int nNumTraces) {

   signal_proc_func_1();

   move_data_1();

   signal_proc_func_2();

   move_data_2();

   .
   .
   .
}
```

Signal processing functions (signal_proc_func_n) would be located in SDRAM and subject to caching, but the UserProc() function itself and data transfer functions (move_data_n) would be located in onchip SRAM (i.e. L2 memory not configured as cache). It is important to remember, *every time* code calls -- or returns from – a function, there is potential for a series of cache misses. In general, it is fair to say that cache is most valuable when code enters a long, multi-index loop and stays there for some time.

4.1.6 Application Code Performance and Architecture Considerations

The analog_io.c file shared by SCI and DSP/BIOS projects holds a function called “ProcessAnalogBuffer()” that contains the basic “real-time loop” which processes buffers of analog I/O data as they become available.

For SCI projects, ProcessAnalogBuffer() is called based on foreground polling in main.c, using the “IsBufferReady() function. For DSP/BIOS projects, IsBufferReady() is called by a DSP/BIOS task that wakes up periodically. This task is instantiated by main.c

When modifying the ProcessAnalogBuffer() function application or performance reasons, here are some suggestions:

- remove DUART char put/get calls to allow faster analog sampling rates
- change rate of LED flashing by varying the counter used (see buf_count variable)

- as faster analog sampling rates are used, it may become necessary to use UART and other peripheral access functions in asynchronous mode. For example ReadUARTData() function can be called in async mode is required

- When using UART I/O, remember that in general you can only read about as many characters as you would expect the baud rate to support during each buffer period. For example, if your buffer size is 100 samples and your sampling rate 20 kHz, then each buffer represents 5 msec. If your baud rate is 19.2 kHz, and you try to read more than about 10 characters, you can cause DUART or DMA controller state machine lock-up, both of which are not good

Initializing Peripheral I/O

To see initialization code examples, look for:

- StandAloneInit() function (standalone.c); this is called from DSP boot code when no host presence is detected (see section 4.4 below, DSP Boot Code, for more information)

- AnalogSetup() function in main.c

- Various peripheral I/O initialization and reset functions in main.c, such as ResetDUART() and InitUART()

4.1.7 CCS Build Output, File Types, and File Locations

CCS generates a ".out" executable (binary) file by default. The .out file conforms to COFF (Common Object File Format) standard -- more or less, with some deviations needed to support Texas Instruments DSPs – see section 4.1.8 below, COFF File Format.

The default Signalogic-provided projects store the resulting DSP executable file on the following subdirectories, depending on hardware type and project type:

Hardware	Build Type	Executable Path	Executable File [1]
SigC54xx PC/104 SigC54xx-PCI DSK C5402	C54xx SCI	\dspcode\c54xx\SCI	tmsc54x.out
	C54xx DSP/BIOS	\dspcode\c54xx\DSP_BIOS	tmsc54x.out
SigC6xxx-PC/104 SigC6xxx PCI DSK C6x11	C6xxx SCI	\dspcode\c6xxx\SCI\	tmsc6x11.out
	C6xxx DSP/BIOS	\dspcode\c6xxx\DSP_BIOS\	tmsc6x11.out
M6x [2]	C6xxx SCI	\dspcode\c6xxx\SCI	tmsiim6x.out

C6xxx DSP/BIOS	\dspcode\c6xxx\DSP_BIOS	tmsiim6x..out

NOTES

[1] .out is the default file extension used by Texas Instruments for COFF files.

[2] A separate executable file is created for M6x hardware as several additional files are included in the M6x board projects that provide support for the numerous I/O modules supported by M6x boards. I/O module examples include A4D4, SD4, SD16, AIX, etc.

Below are some additional notes about CCS build output, file types, and file locations:

1) Project files are located on ..\SCI and ..\DSP_BIOS subdirs.

2) Generated .obj, .asm, and .lst files are located on ..\SCI\cStandard and ..\DSP_BIOS\cStandard subfolders.

3) DSP/BIOS-generated files (.cbd, .s62, .h62, etc) are contained in ..\DSP_BIOS subfolders only.

4.1.8 COFF File Format

Executable (binary) files are produced by CCS in COFF format. COFF is a partial standard used widely, for example by development tools and IDEs offered by Microsoft, Borland, MetroWorks, and others.

If the CCS project options are set for debug mode (-g option specifies “full debug”), then the COFF file will contain symbol table, line number, and other non-executable code information required for symbolic debug. For more information about COFF file formats, please refer to the following TI documents:

TMS320C6000 Assembly Language Tools User’s Guide (Rev M)
Appendix A Common Object File Format
http://focus.ti.com/lit/ug/spru186m/spru186m.pdf

TMS320C54x Assembly Language Tools User’s Guide (Rev F)
Appendix A Common Object File Format
http://focus.ti.com/lit/ug/spru102f/spru102f.pdf

TMS320C55x Assembly Language Tools User’s Guide (Rev G)
Appendix A Common Object File Format
http://focus.ti.com/lit/ug/spru280g/spru280g.pdf

 Copyright © Signalogic 2003-2005

4.2 Host Mode vs. Stand-Alone Mode

When a new DSP code build is created using CCS development tools from the SCI or DSP_BIOS projects, the resulting executable (binary) DSP code can be run, tested, debugged in one of two (modes):

- Host Mode

…or…

- Stand-Alone Mode

These two modes allow SigC5xxx and SigC6xxx hardware to provide flexible configuration options for running in a variety of development and production situations. General categories for configuration include:

- Board connected to a PC bus, under control of or being monitored by, host software programs such as Hypersignal-Macro or application software using DirectDSP software to interface to the hardware (e.g. Visual C/C++, Visual Basic, or MATLAB)
- Board installed in a stand-alone system not connected to a PC, with DSP code booted and run from Flash EEPROM on the board
- some combination of the above

Below is a table that gives some idea of possible hardware configurations, and what modes of operation applies to each.

Configuration	Host PC	JTAG Emulator	Flash EEPROM	Mode	Comments
Development [1]	**X**	**X**	**X**	**Host or Stand-Alone**	Hypersignal-Macro software DirectDSP software application Test programs and utilities
Production, with host PC software GUI	**X**		**X**	**Host**	Application level GUI and software typically uses DirectDSP software for hardware interface; i.e. Visual C/C++, Visual Basic, or MATLAB

Production, enclosure with or without self-contained user-interface [2]			**X**	**Stand-Alone**	DSP code booted from Flash EEPROM on PC104 or PTMC board (or on DSP module in special cases) Host interface possible with USB or RS-232
Field Upgrade or Debug		**X**	**X**	**Stand-Alone**	JTAG emulator used for DSP access Also possible to run standard host software tools by connecting to PC bus using PC bus-to-board adapter cable

Notes

[1] Board is typically installed in a PC (board inserted into PC or connected via passive carrier card or adapter), JTAG emulator is connected to debug header on the board

[2] Board is typically installed in a PC104 stack or other standard external enclosure containing power supply, ac/dc adapter connection, and possible battery backup.

Auto-Detection of Host Mode vs. Stand-Alone Mode

Host Mode vs. Stand-Alone Mode detection is automatic. The SCI boot code in both the C5xxx and C6xxx projects will detect the presence of a host and adjust boot parameters accordingly. This makes Stand-Alone vs. Host mode detection transparent to application DSP software or host software, performed by DSP low-level boot code at run-time, each time the DSP is reset or powered-up.

Stand-Alone operation is defined as whenever no host software is detected; i.e. if the DSP is run without being initialized by Hypersignal or application level software that uses DirectDSP software, such as MATLAB application, Visual C/C++ or Visual Basic GUI, or other user-defined GUI. Note that this definition carefully allows the case where the board is inserted into a PC, but no host software is active – in that case, the boot code will enable Stand-Alone mode.

Initiating Stand-Alone Mode

In Stand-Alone Mode, no host software such as Hypersignal, DirectDSP, etc. is active. There are two (2) main ways that Stand-Alone operation is typically initiated:

1) Download executable DSP code using Code Composer Studio (CCS) software, via JTAG header on the board. Recommended JTAG emulators include Spectrum Digital XDS510PP and XDS510PP-Plus, and SEED XDSpp. The emulator should be connected to the parallel port on the PC, and the parallel port should in EPP mode in the PC's BIOS setup. See section 2.1.5 above, Installing JTAG Emulator Software.

2) Boot from Flash EEPROM (requires JP14 to be installed, and valid program to be contained in EEPROM -- see comments below and section 10.7, Boot/Flash EEPROM). JTAG emulator may or may not be installed and/or active. No host software is active.

Note that in both cases above, the board can either be installed in a PC bus slot, or installed in a separate, non-PC enclosure, such as PC104 stack.

Development configuration is largely a matter of engineer preference. Some engineers prefer to use only CCS tools, performing all code download and debug via JTAG emulation. Other engineers prefer to avoid JTAG completely because it can be slow and tedious, and rely on host software tools to download executable DSP code and "see" algorithm results as the code is running in real-time. Combinations of these modes are also possible.

Stand-Alone Mode Hardware Issues

At the hardware level, Stand-Alone Mode is further configurable to control boot mode and state of processor reset after power-up, controlled by a board jumper, JP14. When installed, JP14 causes the following actions inside the logic controller on the PC104 or PTMC board:

- Take DSPs out of reset and place in idle mode
- Search for valid DSP executable program stored in Flash EEPROM on the PC104 or PTMC board
- Boot and run the program in Flash EEPROM (if found)
- Initialize LED status on the board to reflect "Hardware Stand-Alone Mode Active" (CR2 is illuminated)

 Copyright © Signalogic 2003-2005

4.3 Incorporating Peripheral I/O Access into Applications

To incorporate DMA controller and peripheral I/O access into arbitrary CCS projects, the following files must be included in the application CCS project:

Source or Object File	Peripheral Group	Comments
Executable		
rtc_periph.obj	D, I, L	Or .c if provided
bsp.obj	D, A, R, M	Or .c if provided; note that group D is referenced because rtc_periph uses a few bits of McBSP general-purpose I/O
analog_io.obj	A	Or .c if provided
sd4_init.obj	A	" "
isr4_sd4.obj	A	" "
ss_init.obj	R, M	" "
ss_comm.obj	R, M	" "
Header		
rtc_periph.h	D, I, L	
rtc_dma_board.h	D, I, L	
ss_comm.h	R, M	
analog_io.h	A	
sd_aud.h	A	
rtc_dsp.h	All	
bsp.h	D, A, R, M	

Legend

A	SigSDn audio module
R	RS-422
D	DMA controller
I	DUART, Routing Logic, Flash EEPROM, T1/E1, Digital I/O
L	LEDs
M	McBSP (general initialization)

 Copyright © Signalogic 2003-2005

Also, the following compiler definitions (under "Preprocessor" in CCS menu Project | Build Options) may need to be set:

Compiler Definition	**Comments**
_NCHANNEL_BUILD_	Required for N-channel operation with SigSDn audio modules
_DIGIO_BUILD_	Enables example digital I/O build
_PANELIO_BUILD_	Required for stand-alone units that use panel I/O such as controls, switches, indicators, etc.
_PERIPH_TEST_	Enables test/debug mode for PC/104 peripheral I/O
_CACHE_ENABLE_	Enables onchip L2 memory cache (use only with DSP/BIOS projects; see section 4.1.5 below)

 Copyright © Signalogic 2003-2005

4.4 DSP Code Memory Map Considerations

When creating application code, please keep the following guidelines in mind:

- For C5xxx processors, make sure code is linked with un-initialized holes from 0x0100 to 0x01ff and from 0x0ff0 to 0x11ff

- For C6xxx processors, make sure code is linked with an un-initialized hole from 0x600 to 0x13ff

The above guidelines are strictly adhered to by Signalogic-provided DSP code examples that work with DirectDSP, Hypersignal, and other Signalogic software utilities and applications. Note that in all cases you can examine the .map file produced by CCS to ensure compliance; in the .map file, you should see un-initialized "holes" that match the regions outlined above.

Below is a more detailed explanation of the usage of the areas outlined above:

Description	**C5xxx Region, Section Name**	**C6xxx Region, Section Name**	**Comments**
Reserved area for hardware chip-level registers	`0x0000 – 0x007f`	`N/A`	For C5xxx, no code should reside here. For C6xxx hardware registers are in higher memory, locations.

Interrupt vectors and reserved memory area	`0x0080 – 0x00ff`	`0x0000 – 0x01ff`
		
		`vectors`
			May be relocated in DSP/BIOS projects.

Host interface driver (also known as "talker program")	`0x0100 - 0x01ff`	`0x0600 - 0x0aff`	***Reserved in Host Mode.*** May be overwritten in stand-alone mode *if absolutely necessary*; see MINCONFIG2 Active Configuration in section 4.1.3
Local copy of shared host-DSP properties (below)	`0x0200 – 0x027f`	`N/A`	***Reserved.*** C54xx only. A copy of shared host-DSP properties is maintained at the 0x200 location to allow overlay of direct addressing (DP page set to 4) and indirect addressing using *(addr) notation
Local variables	`0x280 – 0x2ff`	`0x1400 –`	***Reserved.*** Static variables used by low-level SCI code;

		`locvar`	not visible to host. For C6xxx, includes stack, heap, .bss, section. See below.
Legacy areas used for "snap-in" digital filtering, other legacy software functions	`0x0300 – 0x03ff`	`N/A`	

Shared host-DSP properties (sometimes referred to in asm source comments as "global variables")..	`0x042f – 0x04ff`	`0x2000 – 0x21ff`
		
		`vars`
			Reserved. Used in both host and stand-alone modes; do not overwrite. Properties are visible to application C code via rtc_dsp.h header file.
			For C54xx, copied to local area above (0x0200) on program start (see above).

Bss (local data area)	`0x0800`	`0x1400 –`
		
		`.stack`
		`.bss`
		`.sysmem`
		`locvar`
			Can be relocated or resized as needed.

DMA controller driver	`0x0ff0 - 0x0fff`	`0x0fc0 - 0x0fff`	***Reserved.*** Hard, fixed area. Do not overwrite *under any circumstances, regardless of Project type of Active Configuration*
Talker program data transfer area	`0x1000 - 0x10ff`	`0x1000 - 0x13ff`	***Reserved in Host Mode.*** Same comments as "talker program" area, above.
Initial boot code stack (may be changed later by C code or DSP/BIOS initialization)	`0x1100 - 0x11ff`	`0x0b00 - 0x0f00`	Overwrite *only if you are absolutely sure of what you are doing* and fully understand the boot sequence and C variable auto-initialization sequence of your application

Even if you are sure the above regions will not be used in your application, it is still recommended to stay consistent with the Signalogic guidelines to allow compatibility with future applications.

 Copyright © Signalogic 2003-2005

4.5 DSP Boot Code

For C5xxx hardware, low-level boot code is contained in the t54beg.asm file, and for C6xxx hardware, in the t67bg.asm file.

Boot code performs the following functions:

- Critical hardware initialization; for example, onchip operating mode or status registers, setup needed for proper access to external memory, onchip serial port reset, GPIO pins that need to be set/cleared to certain values (such as reset pin for a peripheral)
- initialize stack
- initialize heap (if needed)
- set C global, const, and static variables
- interrogate the system and to determine and initialize basic system flags needed by higher level software, such as processor type, clock rate, host mode vs. stand-alone mode, etc
- SCI variable initialization and host property initialization. In stand-alone mode, the function StandAloneInit() is called to set host properties that would otherwise be left un-initialized. See sections 4.1.6, Application Code Architecture Considerations, and 6.1.2, UserProc: Adding C Code to Real-Time DSP Processing, for more information

C5xxx Processor Notes

1) Boot code in t54beg.asm first sets PMST to 0x00e0, and then set back to 0x00a8 if the processor type auto-detection code finds a C5416 or C542x processor.

4.6 Integrating Off-the-Shelf Algorithms

This section provides assistance in integration of off-the-shelf, real-time DSP algorithms provided by Signalogic into arbitrary systems. The example below uses a LBR (low bitrate) speech codec as a specific example, although the concepts are applicable to a general class of "code / decode" algorithms. Normally this class of algorithms contain the following basic function calls:

- Alg_Init()
- Alg_Encode()
- Alg_Decode()

where "Alg" is a generic algorithm name.

After executing low-level, hardware-related boot code (section 4.5 above, DSP Boot Code), eventually the software will jump to void main(). In the source code excerpts below, sections of an example void main() function are highlighted to show areas relevant to integration into arbitrary DSP systems, as follows:

Hardware-related initialization

Algorithm-related initialization

User-Interface or other user hardware initialization (if applicable)

Analog I/O buffer wait, read, write, and update

Algorithm initialization, encode and decode

Note that the analog I/O buffer code forms an "indefinite while-loop" for real-time processing.

```
void main() {

#ifdef _PERIPH_IO_BUILD_
unsigned int clk_ctrl, stream_ctrl, port_ctrl;
#endif

   if (fStandAloneMode) {
   /*
      Following code is active only if fStandAloneFlag is set in boot code.

      Normally, the properties below are set by VDS software or other
      host software.  In stand-alone mode, some values must be set by the
      DSP code to allow correct operation.

      Note also that many properties are assumed to already have correct
```

```
      default values; see static declarations above.
   */


#ifdef _PERIPH_IO_BUILD_

/* Disable all I/O peripherals and release enable hold, if any -- note this
   is good idea when using JTAG debug, because enable states after JTAG
   stop are unpredictable

*/

   SetDevice(RTC_DEV_ALL, RTC_DEV_ENABLE_RELEASE | DEV_DISABLE);

#endif


   /* Set operating mode to REAL_TIME.  SIMULATION mode valid only for
      presence of host software */

      opmode = REAL_TIME;

   /* Set audio_loopback to 1 for audio testing mode with no algorithm
      code running */

      audio_loopback = 0;

   /* Default melp mode is analysis + synthesis (full-duplex); can be changed
      to ANALYSIS (analysis only), SYNTHESIS (synthesis only); also transcode
      attributes can be added.  See codec.h for possible mode values.*/

      melpmode = ANA_SYN;  // ANA_SYN equivalent to ANALYSIS | SYNTHESIS

      lBaudRate = 1200;

   //   lBaudRate = 2400;   /* uncomment to run MELPe at 2400 bps */

   if (lBaudRate == 1200) {

   /* These are necessary settings to running MELPe at 1200 bps */

      melpmode |= TRANSCODE_DN;
      melpmode |= TRANSCODE_UP;
   }

   /* Set NumFrames to zero to cause auto-initialization of NumFrames and
      NumPackets values.  See melp_rt.c code below and ss_init.c code for
      more comments */

      NumFrames = 0;



      #ifdef _PERIPH_IO_BUILD_

   /* enable Routing Logic internal clock generation and set clock rate */

      clk_ctrl = ReadRoutingLogic(RTC_TDM_RTSW_CLOCK_CTRL);  /* read clock control register */

      clk_ctrl &= ~(RTC_TDM_RTSW_CLK_MODE | RTC_TDM_RTSW_FSYNC);  /* set clock mode to “codec”,
                                                                      clear framesync bits */

      if (sync_serial_enable == SS_PACKET_TRANSFER_MODE)

        clk_ctrl |= RTC_TDM_RTSW_CLK_GEN_EN | RTC_TDM_RTSW_FSYNC_8BIT;  /* enable logic clock,
                                                                            8-bit framesync */
      else

        clk_ctrl &= ~(RTC_TDM_RTSW_CLK_GEN_EN | RTC_TDM_RTSW_FSYNC);  /* disable logic clock,
                                                                          framesync */
```

```
/* internal clock generation setup */

   clk_ctrl &= ~RTC_TDM_RTSW_CLK;  /* first clear clock rate bits... */

   switch (lBaudRate) {            /* then select desired baud rate */

      case 1200:

         if (fUseBitPacking) clk_ctrl |= RTC_TDM_RTSW_CLK_1200;
         else clk_ctrl |= RTC_TDM_RTSW_CLK_2000;

         break;

      case 2000:

      /* this actually causes 1659 Hz rate until CLOCK_PRD register in Routing Logic is
         working */

         clk_ctrl |= RTC_TDM_RTSW_CLK_2000;
         break;

      case 2400:

         if (fUseBitPacking) clk_ctrl |= RTC_TDM_RTSW_CLK_2400;
         else clk_ctrl |= RTC_TDM_RTSW_CLK_3200;

         break;

      case 2700:
         clk_ctrl |= RTC_TDM_RTSW_CLK_2700;
         break;

      case 3200:
         clk_ctrl |= RTC_TDM_RTSW_CLK_3200;
         break;

      case 4000:
         clk_ctrl |= RTC_TDM_RTSW_CLK_4000;
         break;

      case 4800:
         clk_ctrl |= RTC_TDM_RTSW_CLK_4800;
         break;

      case 9600:
         clk_ctrl |= RTC_TDM_RTSW_CLK_9600;
         break;
   }

   WriteRoutingLogic(RTC_TDM_RTSW_CLOCK_CTRL, clk_ctrl);   /* write clock ctrl register */

/* RS-422 port A setup:

   -set clock/framesync direction and clock source

   -set framesync pulse count (none, 8, 16, or 128)
*/

/* set clock direction and source for TDM streams A and B */

   stream_ctrl = ReadRoutingLogic(RTC_TDM_RTSW_STREAM_CTRL);  /* read stream ctrl register */

/* set all clock directions to external (from DSP perspective) */

   stream_ctrl &= ~(RTC_TDM_RTSW_A_CLK_DIR | RTC_TDM_RTSW_A_CLK_DIR | RTC_TDM_RTSW_A_CLK_DIR);

   stream_ctrl |= RTC_TDM_RTSW_B_CLK_SRC;  /* set stream B clock source for internal clock */

   WriteRoutingLogic(RTC_TDM_RTSW_STREAM_CTRL, stream_ctrl);  /* write stream ctrl register */
```

```
  /* set clock direction and source for RS-422 port A */

     port_ctrl = ReadRoutingLogic(RTC_TDM_RTSW_PORT_CTRL);   /* read port ctrl register */

     port_ctrl &= ~(RTC_TDM_RTSW_RSB_CLK_DIR | RTC_TDM_RTSW_RSC_CLK_DIR |
                    RTC_TDM_RTSW_RSD_CLK_DIR);  /* set ports B-D to external clock */

     port_ctrl |= RTC_TDM_RTSW_RSA_CLK_DIR;  /* set RS-422 port A clock direction to internal */

  /* set RS-422 port A clock source to on-logic clock (applies when clock dir is internal) */

     port_ctrl |= RTC_TDM_RTSW_RSA_CLK_SRC;

     WriteRoutingLogic(RTC_TDM_RTSW_PORT_CTRL, port_ctrl);  /* write port ctrl register */


     #endif  /* _PERIPH_IO_BUILD_ */

     #if defined(_PERIPH_IO_BUILD_) && defined (_DIGIO_BUILD_)

    /*
      Enable digital I/O and put a hold on enable state, to avoid Read/Write digital I/O calls
      below doing each time; note that this is not a problem unless other peripherals need to
      be accessed
    */

     SetDevice(RTC_DEV_DIGITAL_IO, RTC_DEV_ENABLE_HOLD | DEV_ENABLE);

     #endif

  }  /* if fStandAloneMode */


/* initialize variables */

   if ((melpmode & 0x0f) == ANA_SYN)
      melpmode = (melpmode & 0xf0) | ANALYSIS | SYNTHESIS;  /* ANA_SYN value still allowed to
                                                               maintain backwards compatibility
                                                               with old "MELP GUI" */

   if (lBaudRate >= 2400) {

      ChSize = CHSIZE_2400;
   }
   else if (lBaudRate <= 2000) {

      ChSize = CHSIZE_1200;
   }

/* Note that host likely will change xxxframeaddr and xxxcoeffaddr
   values below depending on frame and buffer sizes.  Default values
   for old MELP GUI software is 0x7000 and 0x7a00; default values for
   VDS software is 0x1f800 and 0x2f800. */

   rdframeptr = getframeaddr;
   wrframeptr = sendframeaddr;
   rdcoeffptr = getcoeffaddr;
   wrcoeffptr = sendcoeffaddr;

   if (!audio_loopback) {

   /* re-initialize sys_memory */

      minit();

   /* Initialize MELP analysis and synthesis */

      MELPe_Encode_Init();
      MELPe_Decode_Init();
   }
```

```
    bzyflg = 0;
    BUFNUM = 0;  /* set DirectDSP shared mem properties also */
    BFNADD = 0;

    buffer_flg = 0;

    real_time_process();
}


void real_time_process() {

Shortword i, j, nf;
Shortword prev_melpmode;
Shortword cng, csize;
int nTranscodeIn, nTranscodeOut;
BOOL fRdChan, fWrChan;
BOOL fAnalysisTrDn, fAnalysisTrUp, fSynthesisTrUp, fSynthesisTrDn;
unsigned int* pChbuf;
unsigned long rt_count;

#ifdef _DIGIO_BUILD_

unsigned int io_count = 0;
unsigned int bank2_io, bank3_io, bank4_io;

#endif


/* prepare to run MELPe coder/decoder on analog I/O signals */

   loopcnt = 0;
   lStats[0] = 0;
   prev_melpmode = melpmode;

   fAnalysisTrDn = (CodecBitRate == 2400) && (melpmode & TRANSCODE_DN);
   fAnalysisTrUp = (CodecBitRate == 1200) && (melpmode & TRANSCODE_UP);

   fSynthesisTrUp = (CodecBitRate == 2400) && (melpmode & TRANSCODE_UP);
   fSynthesisTrDn = (CodecBitRate == 1200) && (melpmode & TRANSCODE_DN);

   NumPackets = 1;

   if (CodecBitRate >= 2400) {

      if (NumFrames == 0)  {  // if not set by host, then initialize
         if (fAnalysisTrDn && fSynthesisTrUp)
            NumFrames = 3;
         else
            NumFrames = 1;
      }

      if (!fAnalysisTrDn && !fSynthesisTrUp) NumPackets = NumFrames;
   }
   else if (CodecBitRate <= 2000) {

      if (NumFrames == 0)  {
         if (fAnalysisTrUp && fSynthesisTrDn)
            NumFrames = 1;
         else
            NumFrames = 3;
      }
      if (!fAnalysisTrUp && !fSynthesisTrDn) NumPackets = NumFrames;
   }

   ChOffset = ProcIdFlag*ChSize*NumFrames;


#ifdef _DIGIO_BUILD_

/* Initialize digital I/O.  This example simulates user I/O switches, controls, and indicators
```

```
   in a stand-alone enclosure. */

/* i/o bank 2 configuration */

   SetDigitalIOControl(RTC_IO_BANK2, 0x18, DIGIO_ENABLE);  /* tri-color status LED uses 2 output
                                                              bits */

/* i/o bank 3 configuration */

   SetDigitalIOControl(RTC_IO_BANK3, 0xf0, DIGIO_ENABLE);  /* dual tri-color sync/activity LEDs
                                                              use 4 output bits */

/* i/o bank 4 configuration */

   SetDigitalIOControl(RTC_IO_BANK4, 0xe0, DIGIO_ENABLE);  /* battery indicator uses 3 output
                                                              bits */

/*
   Enable digital I/O and put a hold on enable state, to allow subsequent digital I/O calls to
   be somewhat faster.  Note that enable-hold should not be activated if other peripherals need
   to be accessed in between digital I/O calls.
*/

   SetDevice(RTC_DEV_DIGITAL_IO, RTC_DEV_ENABLE_HOLD | DEV_ENABLE);

#endif


/* initialize sync serial communication with external device */

   if (sync_serial_enable != 0) {

      SyncSerialInit(MCBSP1, 0);  // enable sync serial operation (McBSP1)
   }


/* initialize SigSD4 audio module codecs, enable analog I/O interrupts */

   SD4Init(MCBSP0, BRINT0_ENABLE | BXINT0_ENABLE);


/* start real-time do-forever loop */

   while (TRUE) {

/*
  wait until buffer flag from ISR changes; also use time spent waiting
  to perform real-time MIPS measurement.  Value of count x approx 15 nsec
  is number of msec of overhead available
*/
      rt_count = 0;

      while (bzyflg == buffer_flg) {
         rt_count++;
      };

      lStats[1] = rt_count;

      if (audio_loopback) {

      /* The strange 2-line thing below is necessary for C5416, which
         is so fast at 160+ MHz, that we can get a buffer rdy
         message (based on RxPtr) and write samples to the Tx buffer
         before the last TxPtr sample from the previous buffer is output.
         This happens because Tx interrupt occurs slightly after Rx
         interrupt. Anything less than 130 MHz is not a problem, nor
         is non-audio loopback mode, because of the processing delay
         prior to SendData() */

         j = TxPTR;
         while (j == TxPTR);
```

 Copyright © Signalogic 2003-2005

```
         GetData(speech_in, FRAMESIZE*NumFrames, buffer_flg);

         for (i=0; i<FRAMESIZE*NumFrames; i++) speech_out[i] = speech_in[i];

         SendData(speech_out, FRAMESIZE*NumFrames, buffer_flg);

         if (sync_serial_enable != 0) fSync = TRUE;
      }
      else {

      /* check to see if analysis is required */

         if (melpmode & ANALYSIS) {

         /* check for a mode change (e.g. push-to-talk) */

            if (prev_melpmode != melpmode) {

               minit();
               MELPe_Encode_Init();
               prev_melpmode = melpmode;
            }

         /* get uncompressed speech data from analog/digital input buffer */

            GetData(speech_in, NumFrames*FRAMESIZE, buffer_flg);

         /* first, perform analysis */

            for (j=0; j<NumFrames; j++) {

               melp_qpar.chptr = &chbuf_ana[ChOffset+j*ChSize];  /* reset pointer to channel
                                                                    buffer */

               MELPe_Encode(&speech_in[j*FRAMESIZE], &melp_par[j], &melp_qpar);
            }

         /* second, check for transcoding cases (c and d below) */

         /* possible cases:

            a) analysis at 1200, write channal at 1200 (3 frames)
            b) analysis at 2400, write channel at 2400 (1 frame)
            c) analysis at 1200, transcode up, write channel at 2400 (3 frames)
            d) analysis at 2400, transcode down, write channel at 1200 (3 frames)
         */

  /* test code -- save chbuf_ana info */

            melp_qpar.chptr = &chbuf_ana[ChOffset];  /* reset pointer to channel buffer */

            if (fAnalysisTrDn)
              transcode_down(&melp_par[0], &melp_qpar, RD_CHAN, WR_CHAN);

            if (fAnalysisTrUp)
              transcode_up(&melp_par[0], &melp_qpar, !RD_CHAN, !WR_CHAN);
         }

      /* read/write channel */

         if (sync_serial_enable != 0) {

            fSync = TRUE;

// inside sync_serial_enable fSync should be TRUE always, unless synthesis is
// enabled and sync fails on GetSerialData()
//            if ((melpmode & 0xf) == ANALYSIS) fSync = TRUE;  // analysis only

         /* read transmission channel */
```

 Copyright © Signalogic 2003-2005

```
   if ((melpmode & SYNTHESIS) || fSynthesisTrDn || fSynthesisTrUp) {

      for (j=0; j<NumPackets; j++) {

         pChbuf = (unsigned int*)&chbuf_syn[ChOffset+j*ChSize];

         fSync &= GetSerialData(pChbuf, ChSize, j == NumPackets-1);  /* any 0 sync
                                                                        result sets sync
                                                                        to FALSE */
      }
   }

   if (!fSync || (Rx_sync_dead_count > 200)) {

      fSync = FALSE;

      cng = 0;  // set cng to zeros..

      for (j=0; j<NumPackets; j++) {
         for (i=0; i<ChSize; i++) chbuf_syn[ChOffset+j*ChSize+i] = cng;
      }
   }

/* transcode only:  after read channel, before write channel */

   if ((melpmode & 0xf) == TRANSCODE_ONLY) {

      if (lBaudRate >= 2400) {  // channel is 2400+, assume input data is 1200

         nTranscodeIn = TRANSCODE_UP;
         nTranscodeOut = TRANSCODE_DN;
      }
      else {  // vice-versa

         nTranscodeIn = TRANSCODE_DN;
         nTranscodeOut = TRANSCODE_UP;
      }

      fRdChan = FALSE;
      fWrChan = FALSE;

      if ((melpmode & nTranscodeIn) != 0) {

         fRdChan = TRUE;
         fWrChan = ((melpmode & nTranscodeOut) != 0);

         if (nTranscodeIn == TRANSCODE_UP)
           transcode_up(&melp_par[0], &melp_qpar, fRdChan, fWrChan);
         else
           transcode_down(&melp_par[0], &melp_qpar, fRdChan, fWrChan);
      }

      if ((melpmode & nTranscodeOut) != 0) {

         if (!fRdChan) fRdChan = TRUE;
         fWrChan = TRUE;

         if (nTranscodeOut == TRANSCODE_UP)
           transcode_down(&melp_par[0], &melp_qpar, fRdChan, fWrChan);
         else
           transcode_up(&melp_par[0], &melp_qpar, fRdChan, fWrChan);
      }
   }

   /* write transmission channel */

   if ((melpmode & ANALYSIS) || fAnalysisTrDn || fAnalysisTrUp) {

   /* if set by host, create bogus NULL packet with only sync bit
      (typically used for debug purposes */
```

 Copyright © Signalogic 2003-2005

```
               for (j=0; j<NumPackets; j++) {

                  pChbuf = (unsigned int*)&chbuf_ana[ChOffset+j*ChSize];

                  if (fNullPacket) CreateNullPacket(pChbuf, ChSize);

                  SendSerialData(pChbuf, ChSize, j == NumPackets-1);
               }
            }
         }
         else {

            if (fAnalysisTrDn || fSynthesisTrUp) {
               csize = CHSIZE_1200;  // use 1200 bps channel data
               nf = 1;
            }
            else {
               csize = ChSize;
               nf = NumFrames;
            }

            for (j=0; j<nf; j++) {

            /* anasyn_loopback set to 1 by VDS software running on host in
               "Algorithm Loopback" mode */

               if (anasyn_loopback) for (i=0; i<csize; i++)
                 chbuf_syn[ChOffset+j*csize+i] = chbuf_ana[ChOffset+j*csize+i];
            }
         }

      /* check to see if synthesis is required */

         if (melpmode & SYNTHESIS) {

         /* check for a mode change (e.g. push-to-talk) */

            if (prev_melpmode != melpmode) {

               minit();
               MELPe_Decode_Init();
               prev_melpmode = melpmode;
            }


         /* possible cases:

            a) read channel at 1200 (3 frames), transcode up, synthesis at 2400
            b) read channel at 2400 (3 frames), transcode down, synthesis at 1200
            c) read channel at 1200 (3 frames), synthesis at 1200
            d) read channel at 2400 (1 frame), synthesis at 2400
         */

         /* first, check for transcoding cases (a and b above) */

            melp_qpar.chptr = &chbuf_syn[ChOffset];  /* reset pointer to short channel buffer */

            if (fSynthesisTrUp)
              transcode_up(&melp_par[0], &melp_qpar, fAnalysisTrDn, WR_CHAN);

            if (fSynthesisTrDn)
              transcode_down(&melp_par[0], &melp_qpar, !RD_CHAN, !WR_CHAN);

   /* test code -- restore chbuf_syn info */

         /* second, perform synthesis */

            for (j=0; j<NumFrames; j++) {

               melp_qpar.chptr = &chbuf_syn[ChOffset+j*ChSize];  /* reset pointers to short
                                                                    channel buffer */
```

```
            MELPe_Decode(&melp_qpar, &melp_par[j], &speech_out[j*FRAMESIZE]);
         }

      /* send re-synthesized speech results to analog/digital output buffer */

         SendData(speech_out, NumFrames*FRAMESIZE, buffer_flg);
      }
   }

   lStats[2] = fSync;


/* update panel I/O */

#ifdef _DIGIO_BUILD_

   bank2_io = ReadDigitalIOData(RTC_IO_BANK2) & 0xe7;  // mask off outputs  -- note that
   bank3_io = ReadDigitalIOData(RTC_IO_BANK3) & 0x0f;  // masking not necessary if output
   bank4_io = ReadDigitalIOData(RTC_IO_BANK4) & 0x1f;  // bits will not be changed


/* update user I/O display with control values */

   if ((loopcnt & 0xf) == 0) {  /* every 16 buffers, change the LED value (for example for
                                   2400 bps, about 360 msec) */

      io_count++;
      io_count &= 3;
   }

/* rotate tri-color Status LED through all colors */

   if (io_count == 0);  // off
   else if (io_count == 1) bank2_io |= 0x08;  // green
   else if (io_count == 2) bank2_io |= 0x10;  // red
   else if (io_count == 3) bank2_io |= 0x18;  // orange (both on)

/*
   gotoxy(10, vofs);
   printf("Bit Packing = %d  ", bank2 & 0x7);
   gotoxy(30, vofs);
   printf("Operating Mode = %d  ", (bank2 & 0xe0) >> 5);
   gotoxy(50, vofs);
   printf("Audio Loopback = %d  ", bank3 & 1);
*/

/* update display with rear panel control values */

/* rotate tri-color LEDs through all colors */

   if (io_count == 0);  // off
   else if (io_count == 1) {
      bank3_io |= 0x10;  // green
      bank3_io |= 0x40;
   }
   else if (io_count == 2) {
      bank3_io |= 0x20;  // red
      bank3_io |= 0x80;
   }
   else if (io_count == 3) {
      bank3_io |= 0x30;  // orange (both on)
      bank3_io |= 0xc0;
   }

/* update control values on simulated panel */
/*
   gotoxy(30, vofs+6);
   printf("Noise PreProcessor = %d  ", (bank3 & 2) >> 1);
   gotoxy(30, vofs+7);
   printf(" Post-Filter = %d  ", (bank3 & 4) >> 2);
```

 Copyright © Signalogic 2003-2005

```
      gotoxy(30, vofs+9);
      printf("Option 1 = %d  ", bank4 & 1);
      gotoxy(50, vofs+9);
      printf("Option 2 = %d  ", (bank4 & 2) >> 1);
      gotoxy(30, vofs+11);
      printf("Option 3 = %d  ", (bank4 & 4) >> 2);
      gotoxy(50, vofs+11);
      printf("Option 4 = %d\n", (bank4 & 8) >> 4);
      gotoxy(50, vofs+12);
      printf("Option 4 = %d\n", (bank4 & 10) >> 5);
*/

   /* update indicators */

      WriteDigitalIOData(RTC_IO_BANK2, bank2_io);
      WriteDigitalIOData(RTC_IO_BANK3, bank3_io);
      WriteDigitalIOData(RTC_IO_BANK4, bank4_io);

#endif


   /* update buffer flags */

      buffer_flg = 1 - buffer_flg;     /* toggle algorithm buffer indicator */

      if ((sync_serial_enable == 0) || fSync) {

      /* increment loop counter (used by host PC to measure packet interval) */

         loopcnt++;
         lStats[0] = loopcnt;

         BUFNUM = buffer_flg;       /* set DirectDSP shared mem properties also */
         BFNADD = buffer_flg;
      }
   }
}
```

 Copyright © Signalogic 2003-2005

5 Building Host Application Software

In this section are some examples of building Visual Studio, .NET, and MATLAB software applications that interface directly to DSP hardware and real-time DSP code, via the DirectDSP software interface.

DirectDSP software is an intermediate library that ties host application level software to DSP hardware. On its "API side", DirectDSP software presents a well-documented, consistent series of function calls to C/C++, Visual Basic, and MATLAB programs. On its "driver side", DirectDSP software interfaces with Win9x, WinXP, or Linux drivers that allow ISA bus, PCI bus, or parallel port access directly to DSP devices. At the hardware level, typically the HPI (host port interface) peripheral on the DSP device(s) is used to connect the host bus to the DSP and its onchip resources, via circuitry and logic on the DSP board.

Following are some examples of functionality provided or allowed by DirectDSP software API calls and its underlying driver interface:

- Access onchip DSP memory, offchip board memory, and registers or FIFOs contained in data acquisition and control modules, telecom modules, and other board peripheral I/O devices

- Download DSP code in COFF file format, as created by Texas Instruments CCS development tools

- Start, stop, and otherwise control DSP processors and installed modules

- Allow host and GUI application software to "see" DSP software data objects without stopping or pausing the DSP processor

- Monitor data and parameters set by DSP code in real-time; for example update graphical displays, gauges, meters, text displays

- Set properties in DSP SCI (Source Code Interface) framework, for instance sampling rate, number of channels, real-time buffer sizes, etc

DirectDSP fully supports multiprocessor hardware, including "broadcast mode" for DSP onchip memory writes, code download and run, device reset.

5.1 Building Visual Studio Software

Under Win9x and WinXP, Visual Studio programs can directly access SigC5xxx, SigC6xxx, and M6x hardware using DirectDSP software. In that case, you can take advantage of a direct PC bus interface to the DSP hardware – a faster, non-instrumented approach that avoids bottlenecks and slowdowns found in RTDX or other JTAG serial debug stream methods.

With DirectDSP software, DSP/BIOS and RTDX components are not necessary in DSP code, or they can be used at the same time. If RTDX is not used, then some amount of MIPS and memory footprint is saved. Via the DirectDSP interface, any variable, any buffer, any piece of data can be examined or transferred at any time, on an asynchronous basis, regardless of what the DSP processor is doing -- even during interrupts. The only requirement is that the "talker" interrupt routine remain resident in DSP memory, the default situation for DSP code provided by Signalogic. For DSP code modified or created using TI development tools, maintaining talker functionality requires linking application code to leave an un-initialized hole for the talker.

Note that when using DirectDSP API functions, the talker program is downloaded, initialized, and run by DSInitBoard() or DSInitProcessor() calls. After one of those calls is made, other DirectDSP functions such as DSGetMem(), DSPutMem(), DSSetDSPProperty(), and others are active at that time. These functions stay active before and after executable DSP code is downloaded, regardless of whether the DSP(s) are running or not.

For C6xxx hardware, the talker program occupies onchip memory from 0x600 to 0x1100 and uses HWI 12); for C54xx hardware the talker program occupies onchip DSP memory from 0x100 to 0x1ff and from 0x1000 to 0x11ff. For more information, see section 4.4 above, DSP Code Memory Map Considerations.

5.1.1 Running Visual C/C++ and Visual Basic Programs with DSP Hardware

To build and run a Visual Studio C/C++ host program example, follow the steps below.

1. First, run Visual Studio v6.0.

   Typically you would select Start → Programs → Microsoft Visual Studio 6.0 → Microsoft Visual C++ 6.0, as shown in Figure 5-1 below.

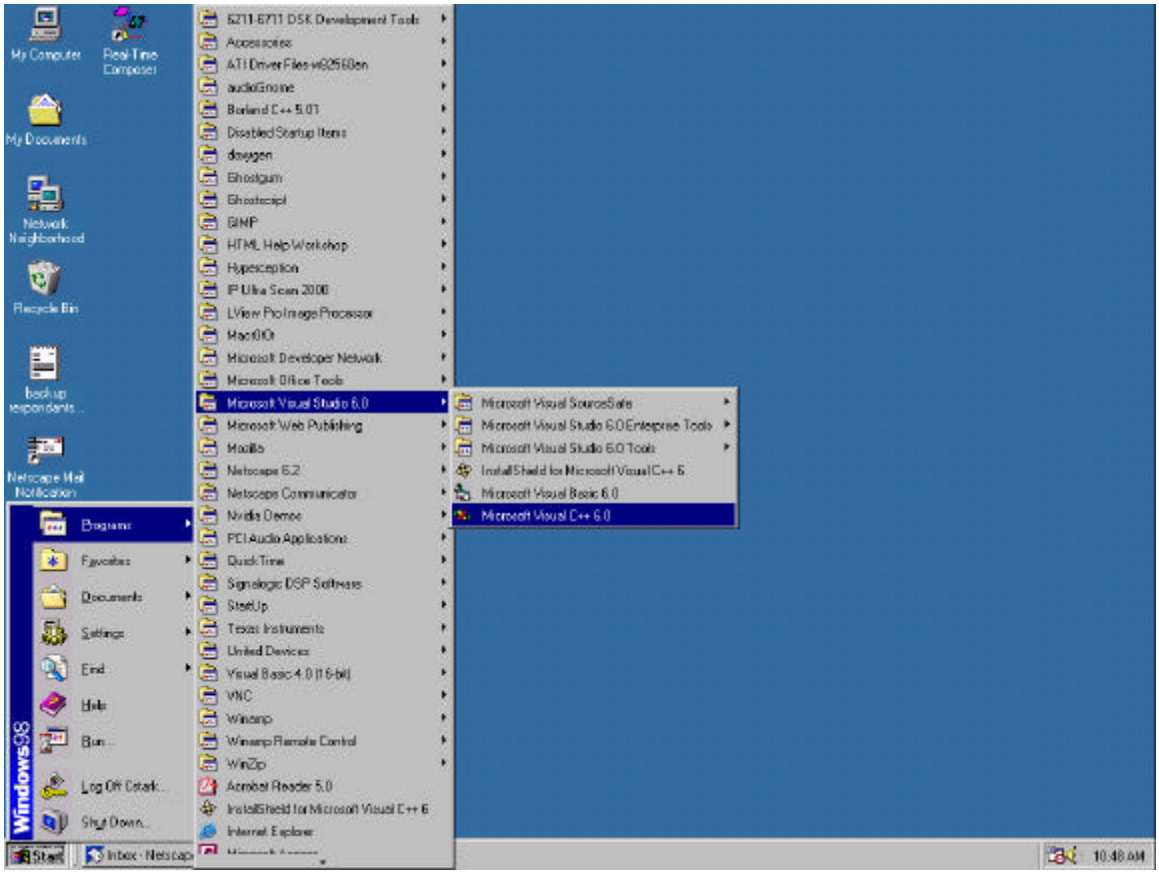


Figure 5-1, Starting Visual C++

2. Load project dscope32ms.dsw or dscope32ms.dsw located at \dspower\hwlib\ into the workspace.
3. Build the project by Build → Build Dscope32ms.exe or F7. If build successfully, will see the outputs as in Figure 5-2 below.

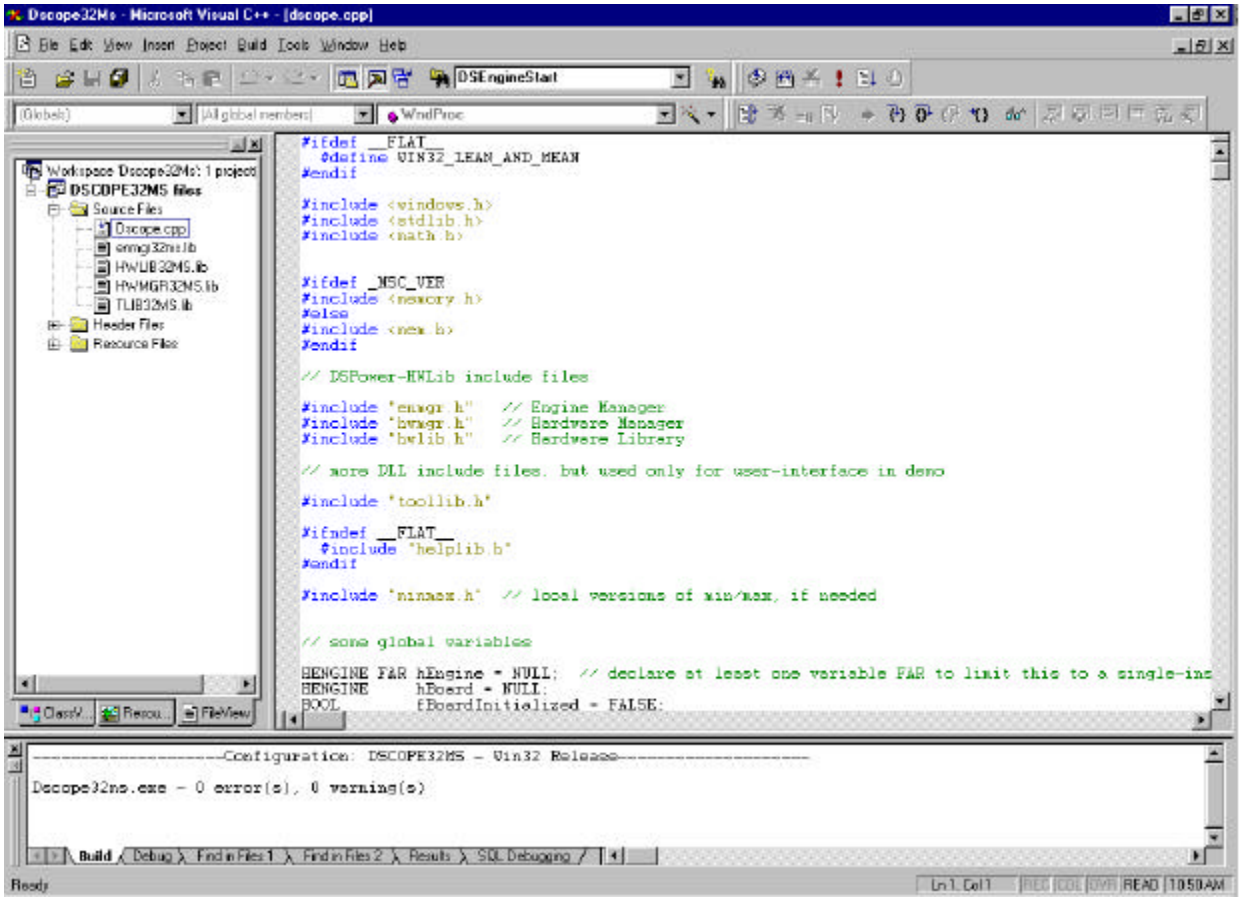


Figure 5-2, Building the Dscope Demo

4. Run the program by Build → Execute Dscope32ms.exe or Ctrl+F5.
5. You will see the Hardware Manager dialog box as in Figure 5-3 below.

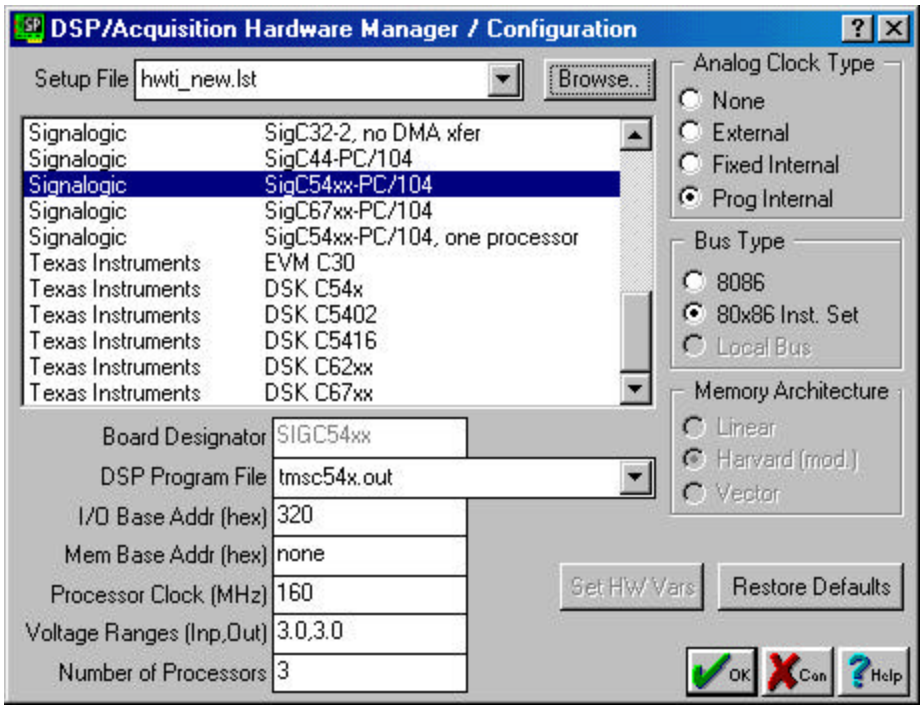


Figure 5-3, Hardware Manager Dialog

6. Choose the proper board type and click Ok, you will see the Digital Scope program running and display some trace. If you don't have any input hooked up to the board, you will just see a flat line.

 Copyright © Signalogic 2003-2005

Figure 5-4, Digital Scope Demo, Initial Display

7. If you click on the left-most button to expand the vertical scale, you will see some noise in the display as in Figure 5-5 below.

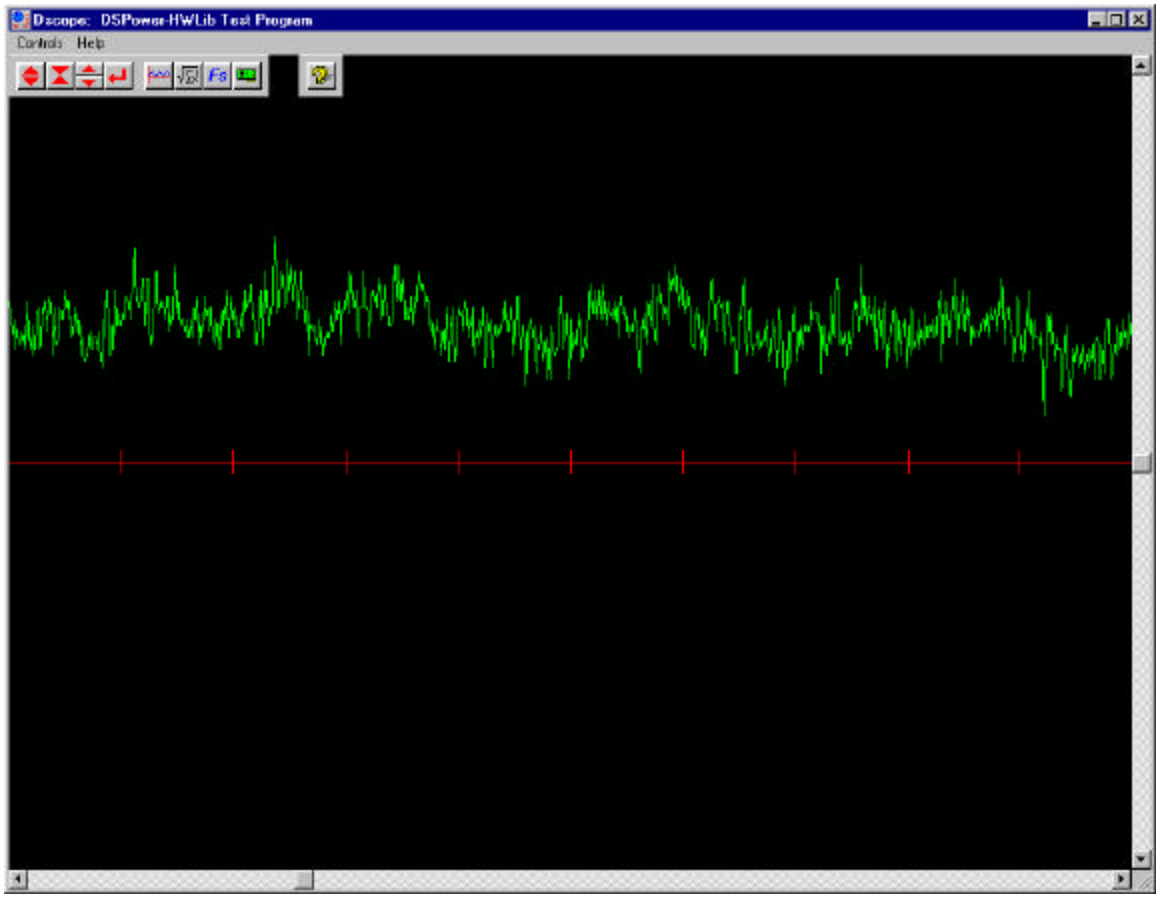


Figure 5-5, Digital Scope with Vertical Scale Expanded to Show Analog Noise

5.1.2 Interface Between Visual C/C++ and Real-Time DSP Code

Below is an example C/C++ program that uses DirectDSP software to download executable DSP code and monitor DSP C code variables and data buffers as the code runs in real-time on the DSP hardware.

Some important actions in the example C/C++ source to look for (highlighted in red text):

- Displaying Hardware Manager dialog, to allow choice of DSP / data acquisition hardware type
- Initialize hardware drivers
- Assigning a DSP board handle
- Initializing a DSP board
- Downloading executable DSP code

- DSGetDSPProperty and DSPutDSPProperty examples, showing how to access default properties in run-time DSP code
- Activating UserProc() function in real-time DSP C code
- DSGetSymbolAddress examples showing how to access user-defined C code symbols
- Reset one or more DSPs
- Running DSP code
- Using DSGetMem to access DSP code symbols and buffers while DSP code is running

Some notes about the above actions:

1) For more information on the UserProc() function, see section 6.1 below, Host Mode and UserProc() Debugging

2) xxxBoard functions have xxxProcessor function counterparts which can be used to specify one or more processors on a multiprocessor board. For example, InitBoard(hBoard) will initialize all processors on a board, InitProcessor(hBoard, 0x7) will initialize only the first three (3) processors. The last parameter in xxxProcessor function calls is used as a "processor list" where each bit specifies a processor; any combination of processors is valid.

Note that comments in the C/C++ code below are highlighted in blue.

```
#include <windows.h>
#include <stdlib.h>

// DirectDSP® software package include files

#include "enmgr.h"   // Engine Manager
#include "hwmgr.h"   // Hardware Manager
#include "hwlib.h"   // Hardware Library


// some global variables

  ...note:  many variable and type declarations omitted for brevity...

HANDLE FAR  hEngine = NULL;  // declare at least one variable FAR to limit this to a
single-instance program
HANDLE      hBoard = NULL;
BOOL        fBoardInitialized = FALSE;
char        szBoard[30];
UINT        uMemArch;
int         wBuflen = 1024;
DWORD       dwMemSize,dwBufferBaseAddr;
HWND        hwndScope = NULL;
WORD        wBoardClass;

float       FsDesired = 22050.0;
float       FsActual;
DWORD       dwFsMode;
short int   ChanList = 0;
```

```
int PASCAL WinMain (HANDLE hInstance, HANDLE hPrevInst, LPSTR lpszCmdParam, int nCmdShow) {

// comment this call if status messages from DirectDSP should not be visible

   DSShowEngMgrStatusWindow();  // turn on debug status window in Engine Manager


/*
   Show hardware selector dialog (Hardware Manager); returns board designator string note:  if
   hardware choice is fixed, then skip this call and use correct board designator string in
   DSAssignBoard call below
*/

   if (DSShowHardwareSelectorDlg(NULL, szBoard) == IDCANCEL) goto cleanup;


/* open dsp engine or driver

  Notes:

  1) Under Win9x, default action is to first attempt to open Hypersignal-Macro (or Macro EX) as
     engine)

  2) Under WinXP or Linux, default action is to open/initialize kernel mode drivers

*/

   hEngine = DSEngineOpen(DS_EO_HSM, NULL);

   if (!hEngine) {

      hEngine = DSEngineOpen(DS_EO_HSA, NULL);

      if (!hEngine) {

         itoa(DSGetHWLibErrorStatus(), tmpstr, 10);
         lstrcat(tmpstr, " is error code; DSEngineOpen failed");
         MessageBox(GetActiveWindow(), tmpstr, "Dscope Test Prog", MB_OK);
         goto cleanup;
      }
   }


// assign a board handle:  engine handle, board designator, bus type, IO base addr, Mem base addr

   hBoard = DSAssignBoard(hEngine, szBoard, NULL, NULL, NULL);

// initialize the board; make sure it's installed, reset all processors

   fBoardInitialized = DSInitBoard(hBoard);

   if (!fBoardInitialized) {

      itoa(DSGetEngineErrorStatus(hEngine), tmpstr, 10);
      lstrcat(tmpstr, " is error code; DSInitBoard failed");
      MessageBox(GetActiveWindow(), tmpstr, "Dscope Test Prog", MB_OK);
      goto cleanup;
   }


// interrogate engine for board type values

   wBoardClass = DSGetBoardClass(hBoard);


// get memory architecture

   uMemArch = DSGetMemArch(hBoard);
```

```
   if (uMemArch == NULL) {

     MessageBox(GetActiveWindow(), "DSGetMemArch failed","Dscope Test Prog", MB_OK);
     goto cleanup;
   }


// determine sampling rate ctrl. reg. value, and actual rate (closest rate possible to desired);
// CalcSampFreq returns ctrl. reg. value directly, uses ptr to return actual sampling frequency
// (in Hz)

   dwFsMode = DSCalcSampFreq(hBoard, FsDesired, 1, &ChanList, &FsActual);  // demo assumes 1
                                                                           // channel, initial
                                                                           // value 22.05 kHz


// load executable DSP code file (usually a COFF file produced by DSP manufacturer's linker)

   if (!DSLoadFileProcessor(hBoard, NULL, 1)) {  // load default file for the board type
                                                 // (processor 0 only)

      MessageBox(GetActiveWindow(), "DSLoadFileBoard:  problem loading file", "Dscope Test Prog",
                 MB_OK);
      goto cleanup;
   }


// get the memory size, (note that this currently has to be done after LoadFile)

   dwMemSize = DSGetMemSize(hBoard, 0x01);  // processor 0 only


// reset the DSP board (should already be in reset state; processor 0 only)

   DSResetProcessor(hBoard, 0x01);


// set relevant SCI properties

   DSPutDSPProperty(hBoard, DSP_BOARDCLASS, wBoardClass & 0x0ff);  // main type in low byte
   DSPutDSPProperty(hBoard, DSP_BOARDSUBCLASS, wBoardClass >> 8);  // subtype in high byte

   DSPutDSPProperty(hBoard, DSP_OPMODE, 2);       // Dig. Scope is mode 2

   DSPutDSPProperty(hBoard, DSP_FILTTYPE1, 0);    // disable trace 1 real-time filter
   DSPutDSPProperty(hBoard, DSP_FILTTYPE2, 0);    // disable trace 2 real-time filter

   DSPutDSPProperty(hBoard, DSP_TRIGLEVEL, 0);    // free-run triggering
   DSPutDSPProperty(hBoard, DSP_TRIGCHANLIST, 0);

   DSPutDSPProperty(hBoard, DSP_BUFLEN, wBuflen); // buffer size

   DSPutDSPProperty(hBoard, DSP_HOSTBUFNUM, 0);
   DSPutDSPProperty(hBoard, DSP_BUFNUM, 0);

   DSPutDSPProperty(hBoard, DSP_CHANLIST, 0);     // starting channel
   DSPutDSPProperty(hBoard, DSP_NUMCHAN, 1);      // number of channels

   DSPutDSPProperty(hBoard, DSP_GAINLIST, 0);     // gain list

   DSPutDSPProperty(hBoard, DSP_FSMODE, dwFsMode);  // sampling rate control register (mode
                                                    value)

   DSPutDSPProperty(hBoard, DSP_FSVALUE, *((long*)&FsActual));  // actual sampling rate (in Hz)


// get address of input time domain data

   dwBufferBaseAddr = DSGetDSPProperty(hBoard, DSP_TIMDATAADDR);
```

```
   if (!hPrevInst) {

  ...note:  window class registration and CreateWindow code omitted for brevity...

      hwndScope = CreateWindow(...

      ShowWindow(hwndScope, nCmdShow);
      UpdateWindow(hwndScope);
   }


// scope window is born; turn the board loose...  (processor 0 only)

   DSRunProcessor(hBoard, 0x01);


// tell engine to wait for a buffer, and then send a BUFRDY message to our window when one is
// ready

   DSRegisterEngineMsgWnd(hEngine, DS_REMW_SETDSPDATARDYMSG, hwndScope);  // register callback
                                                                          // window for buffer
                                                                          // ready messages

   DSWaitForBuffer(hBoard, 0, NULL, DS_WFB_POLLED);


// time to give up and become a Windows animal

   while (GetMessage(&msg, NULL, 0, 0)) {

      TranslateMessage (&msg);
      DispatchMessage (&msg);
   }


cleanup:

   if (fBoardInitialized) DSDisableBoard(hBoard);  // disable board (all processors)

   if (hBoard != NULL) DSFreeBoard(hBoard);

   if (hEngine != NULL) DSEngineClose(hEngine);

   return msg.wParam;
}


long WINAPI _export WndProc(HWND hwnd, WORD wMsg, WORD wParam, LONG lParam) {

    ...note:  most variable declarations omitted for brevity...

static HGLOBAL    hBuf = NULL;
static int far*   buf;
static short int  nCurBuf = 0;

   switch (wMsg) {

   ...note:  other message-processing code omitted for brevity...

      case WM_CREATE:

      // allocate some memory to hold board buffers

         hBuf = GlobalAlloc(GMEM_ZEROINIT | GMEM_MOVEABLE | GMEM_DISCARDABLE,
                            MAX_BUF_LEN*sizeof(int));

         break;


      case WM_DSPENGINE_BUFRDY:  // message sent by DSP engine when data buffer is ready
```

```
         buf = (int far*)GlobalLock(hBuf);

      // transfer below illustrates two common type of DSP memory architectures

         if (uMemArch == DS_GMA_VECTOR) {  // vector data memory

            if (nCurBuf == 0)
               uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_X, dwBufferBaseAddr, DS_GM_SIZE16,
                                  buf, wBuflen);
            else
               uStatus = DSGetMem(hBoard, DS_GM_VECTOR_DATA_Y, dwBufferBaseAddr, DS_GM_SIZE16,
                                  buf, wBuflen);

         }
         else { // linear data/prog memory, or modified harvard arch. with linear data memory

            uStatus = DSGetMem(hBoard, DS_GM_LINEAR_DATA_RT, dwBufferBaseAddr+nCurBuf*wBuflen,
                               DS_GM_SIZE16, buf, wBuflen);
         }

         GlobalUnlock(hBuf);

         if (!uStatus) MessageBox(hwnd, "DSGetMem:  problem with point transfer", "Dscope Test
                                  Prog", MB_OK);
         else FirstBufferRcvd = TRUE;


         nCurBuf ^= 1;  // switch buffers

         DSPutDSPProperty(hBoard, DSP_HOSTBUFNUM, nCurBuf);  // write the new buffer #

         if (!IsIconic(hwnd)) DSWaitForBuffer(hBoard, nCurBuf, NULL, DS_WFB_POLLED);

         break;

   }

   return 0L;
}
```

 Copyright © Signalogic 2003-2005

5.2 Building MATLAB Software

Under Win9x and WinXP, MATLAB and related software can directly access SigC5xxx, SigC6xxx, and M6x hardware using DirectDSP software. From MATLAB, SIMULINK, or Real-Time Workshop software, it is possible to call DirectDSP API functions from .m code. In that case, you can take advantage of a direct PC bus interface to the DSP hardware – a faster, non-instrumented approach that avoids bottlenecks and slowdowns found in RTDX or other JTAG serial debug stream methods.

With DirectDSP software, DSP/BIOS and RTDX components are not necessary in DSP code, or they can be used at the same time. If RTDX is not used, then some amount of MIPS and memory footprint is saved. Via the DirectDSP interface, any variable, any buffer, any piece of data can be examined or transferred at any time, on an asynchronous basis, regardless of what the DSP processor is doing -- even during interrupts. The only requirement is that the "talker" interrupt routine remain resident in DSP memory, the default situation for DSP code provided by Signalogic. For DSP code modified or created using TI development tools, maintaining talker functionality requires linking application code to leave an un-initialized hole for the talker.

Note that when using DirectDSP API functions, the talker program is downloaded, initialized, and run by InitBoard() or InitProcessor() calls. After one of those calls is made, other DirectDSP functions such as GetMem(), PutMem(), SetDSPProperty(), and others are active at that time. These functions stay active before and after executable DSP code is downloaded, regardless of whether the DSP(s) are running or not.

For C6xxx hardware, the talker program occupies onchip memory from 0x600 to 0x1100 and uses HWI 12); for C54xx hardware the talker program occupies onchip DSP memory from 0x100 to 0x1ff and from 0x1000 to 0x11ff. For more information, see section 4.4 above, DSP Code Memory Map Considerations.

5.2.1 Running MATLAB Programs with DSP Hardware

First, run MATLAB, any version 5.0 or higher.

1. When the MATLAB Command Window appears, type in proper directory for the MATLAB examples as shown in Figure 5-6 below, MATLAB Command Window.

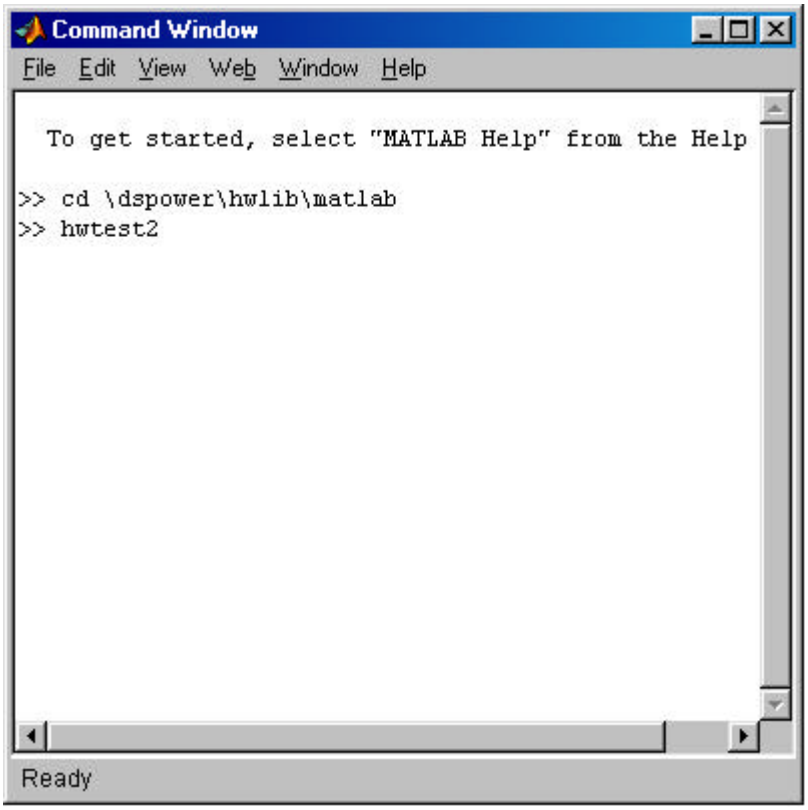


Figure 5-6, MATLAB Command Window

2. For example: Running the hwtest2.m test program as shown in Figure 5-6 above will bring up a Hardware Manager dialog as in Figure 5-3 above.
3. Choose the proper board type and click Ok. If you don't have any input hooked up such as function generator, you will see signal noise as shown in Figure 5-7 below.

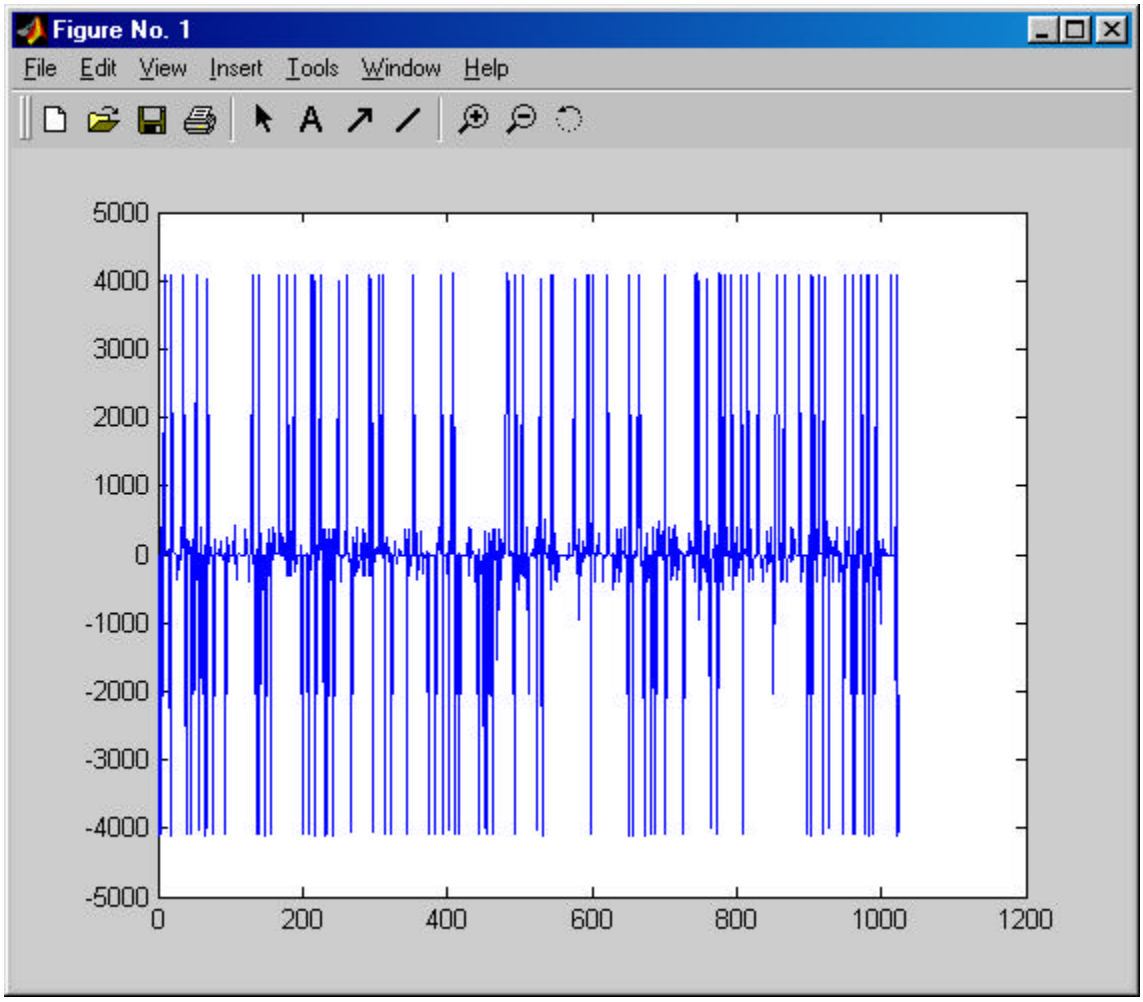


Figure 5-7, MATLAB Figure Window

5.2.2 Interface Between MATLAB and Real-Time DSP Code

Below is an example .m program that uses DirectDSP software to download executable DSP code and monitor DSP C code variables and data buffers as the code runs in real-time on the DSP hardware.

Some important actions in the example .m source to look for (highlighted in red text):

- Adding important subfolders to MATLAB search path
- Displaying Hardware Manager dialog, to allow choice of DSP / data acquisition hardware type
- Initialize hardware drivers
- Assigning a DSP board handle

- Initializing a DSP board
- Downloading executable DSP code
- GetDSPProperty and PutDSPProperty examples, showing how to access default properties in run-time DSP code
- Activating UserProc() function in real-time DSP C code
- GetSymbolAddress examples showing how to access user-defined C code symbols
- Running DSP code
- Using GetMem to access DSP code symbols and buffers while DSP code is running

Some notes about the above actions:

1) For more information on the UserProc() function, see section 6.1 below, Host Mode and UserProc() Debugging

2) xxxBoard functions have xxxProcessor function counterparts which can be used to specify one or more processors on a multiprocessor board. For example, InitBoard(hBoard) will initialize all processors on a board, InitProcessor(hBoard, 0x7) will initialize only the first three (3) processors. The last parameter in xxxProcessor function calls is used as a "processor list" where each bit specifies a processor; any combination of processors is valid.

```
% hwtest6.m
% test / example MATLAB program for DirectDSP software
%
% shows hardware selector, opens DSP engine,
% initializes selected board, sends buffer to analog output (D/A), and
% inputs an analog input (A/D); i.e. performs continuous analog input/output
% buffering with the DSP/data acq. board
% yl, 5 May 1999
% sh, 28 May 1999
% jhb, 29 May 1999
% rs, jhb 12 Feb 2002

% IMPORTANT NOTE:
%
% MATLAB v4.2x appears to have a bug where using the Close button (upper left
% of command window) does not correctly free DLLs that have been used by
% MATLAB programs.
%
% Use the "exit" command instead.
%
% Otherwise, when using DirectDSP API calls in a subsequent MATLAB
% session, you may experience GPF or 'illegal operation' errors, or be unable
% to run other, non-MATLAB programs that use the hwlib.dll, hwlib32.dll,
% hwmgr.dll, or hwmgr32.dll programs.

% determine MATLAB version, display notes and warnings if necessary.

MatVer=GetMVer(1);

% Below are important subdirectories to add to MATLAB's search path; modify
% these if Windows and/or DirectDSP software were not installed using the default
```

 Copyright © Signalogic 2003-2005

```
% root names:

fid=fopen('\windows\win.ini');
if (fid ~= -1)
    path(path, '\windows\system');
else
    fid=fopen('c:\windows\win.ini');
    if (fid ~= -1)
        path(path, 'c:\windows\system');
    else
        fid =fopen('d:\windows\win.ini');
        if (fid ~= -1)
            path(path, 'd:\windows\system');
        end
    end
end
if (fid ~= -1)
    fclose(fid);
end

fid=fopen('\dspower\hwlib\hwsetup.lst');
if (fid ~= -1)
    path(path, '\dspower\hwlib');
else
    fid=fopen('c:\dspower\hwlib\hwsetup.lst');
    if (fid ~= -1)
        path(path, 'c:\dspower\hwlib');
    else
        fid=fopen('d:\dspower\hwlib\hwsetup.lst');
        if (fid ~= -1)
            path(path, 'd:\dspower\hwlib');
        end
    end
end
if (fid ~= -1)
    fclose(fid);
end

% load constants needed by test program; note that v5 constants are declared
% "global" because v5 load function is so pitifully slow, and calling load
% inside individual .m functions is unworkable

if (hex2num(MatVer) < hex2num('500'))

    load windows.con -mat  % misc. constants used by DSPower test and example programs
    load enmgr.con -mat    % Engine Manager constants
    load hwmgr.con -mat    % Hardware Manager constants
    load hwlib.con -mat    % Hardware Library constants

else
    load windows32.con -mat  % Win32 versions
    load enmgr32.con -mat
    load hwmgr32.con -mat
    load hwlib32.con -mat
end


% initialize flags

fCanceled = FALSE;          % result of hardware selector dialog
hEngine = NULL;             % engine handle
hBoard = NULL;              % board handle
fBoardInitialized = FALSE;  % board status

ShowStatusWindow;           % turn on debug status window in Engine Manager    *****

% show hardware selector dialog box ("Hardware Manager"); returns board designator
% string selected by user
% note:  if hardware choice is fixed, then skip this call and use correct board designator
% string in AssignBoard call below
```

 Copyright © Signalogic 2003-2005

```
  [fSelect, BoardStr] = ShowHwSelector(NULL);

 if (fSelect == IDCANCEL)  % user canceled?
  return;
 end


% open dsp engine or driver
%
% Notes:
%
%1) Under Win9x, default action is to first attempt to to open Hypersignal-Macro (or Macro EX) as
%   engine)
%
% 2) Under WinXP or Linux, default action is to open/initialize kernel mode drivers


hEngine = EngineOpen(DS_EO_HSM, NULL, NULL, DS_EO_NOTVISIBLE);

if (hEngine == NULL)

    hEngine = EngineOpen(DS_EO_HSA, NULL, NULL, DS_EO_NOTVISIBLE);  % if that doesn't work, try
Hypersignal-Acoustic

    if (hEngine == NULL)

        errstr = ['EngineOpen failed; error code = ' num2str(GetHwLibErrStat)];
        disp(errstr);

        return;  % abort
    end
end


% assign a board handle:  engine handle, board designator string, instruction set, IO base addr,
% Mem base addr

hBoard = AssignBoard(hEngine, BoardStr, NULL, NULL, NULL);  % NULL == use defaults where possible


% initialize the board; make sure it's installed and responding

fBoardInitialized = InitBoard(hBoard);

if (fBoardInitialized == FALSE)

    errstr = ['InitBoard failed; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);

    if (hBoard ~= NULL)
        FreeBoard(hBoard);
    end

    if (hEngine ~= NULL)
        EngineClose(hEngine);
    end

    return;  % abort
end


% load executable DSP code file (usually a COFF file produced by DSP manufacturer's linker)

if (LoadProcessor(hBoard, NULL, 1) == NULL)  % load default file for the board type (processor 0
                                             % only)

    errstr = ['LoadProcessor failed; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);

    if (hBoard ~= NULL)
        FreeBoard(hBoard);
```

 Copyright © Signalogic 2003-2005

```
    end

    if (hEngine ~= NULL)
        EngineClose(hEngine);
    end

    return;  % abort
end


% prepare hardware and data transfer values, allocate memory to receive buffers

BufNum = 0;               % start with buffer 0
NumChan = 1;              % this example uses one channel
InputOffset  = 0.0;       % DSP input offset
OutputOffset = 0.0;       % DSP output offset
InputScaleFactor  = 1.0;  % DSP input scale factor
OutputScaleFactor = 1.0;  % DSP output scale factor
desiredSampFreq = 20000;  % desired sampling rate
desiredOutFreq = 1000;    % desired output frequency

BufSize = min(desiredSampFreq*5, 4096);  % default buffer size

BoardClass = GetClass(hBoard);       % board classification
MemArch = GetMemArch(hBoard);        % board memory architecture

if (MemArch == DS_GMA_VECTOR)
 BufMul = 1;                         % multiplier for BufSize
else
 BufMul = 2;
end

TimBaseAddr = GetDSPProperty(hBoard, DSP_TIMDATAADDR);  % base address of input
OutBaseAddr = TimBaseAddr+BufMul*BufSize;          % base address of output

[FsMode, FsActual] = CalcSampFreq(hBoard, desiredSampFreq, NumChan, NULL);

buf = zeros(1, BufSize);       % allocate buffer memory  ***
buf1 = zeros(1, BufSize);
buf2 = zeros(1, BufSize);
buf_plot = zeros(1, BufSize);

ResetBoard(hBoard);  % reset board (should already be in reset state)

% write property values to DSP code on board

PutDSPProperty(hBoard, DSP_BOARDCLASS, rem(BoardClass, 256));    % main type in low byte
PutDSPProperty(hBoard, DSP_BOARDSUBCLASS, fix(BoardClass/256));  % subtype in high byte

PutDSPProperty(hBoard, DSP_OPMODE, 6);          % continuous input/output mode

PutDSPProperty(hBoard, DSP_FILTTYPE1, 0);       % disable trace 1 real-time filter
PutDSPProperty(hBoard, DSP_FILTTYPE2, 0);       % disable trace 2 real-time filter

PutDSPProperty(hBoard, DSP_BUFLEN, BufSize);    % input buffer size
PutDSPProperty(hBoard, DSP_STMBUFLEN, BufSize); % output buffer size

PutDSPProperty(hBoard, DSP_STMDATAADDR, OutBaseAddr);  % output buffer base address

PutDSPProperty(hBoard, DSP_HOSTBUFNUM, 0);      % host buffer number
PutDSPProperty(hBoard, DSP_BUFNUM, 0);          % DSP buffer number

% **** USE CHANNEL 2 FOR IAI

PutDSPProperty(hBoard, DSP_CHANLIST, 2);        % channel list (each channel is 4 bits, start
from left; first channel is 0)
PutDSPProperty(hBoard, DSP_NUMCHAN, NumChan);   % number of channels

PutDSPProperty(hBoard, DSP_GAINLIST, 0);        % gain/attenuation list

PutDSPProperty(hBoard, DSP_TRIGLEVEL, 0);          % free-run triggering  ***
```

```
PutDSPProperty(hBoard, DSP_TRIGCHANLIST, 0);       % triger channel  ***

% set output digital scale (multiply) factor and offset (add) to DSP code

if (GetBoardInfo(hBoard, DS_GBI_DSPWORDLENGTH) == 24)

    nShift = 7;    % don't ask; weird efficiency thing required by DSP56xxx boards
    nShift2 = 8;
else
    nShift = 0;
    nShift2 = 0;
end

PutDSPProperty(hBoard, DSP_SCALEOUT,  256*OutputScaleFactor*(2^nShift));  % scale factor is
implemented Q-8 in DSP code
PutDSPProperty(hBoard, DSP_SCALEIN,   256*InputScaleFactor*(2^nShift));
PutDSPProperty(hBoard, DSP_OFFSETOUT, OutputOffset*(2^nShift2));
PutDSPProperty(hBoard, DSP_OFFSETIN,  InputOffset);

PutDSPProperty(hBoard, DSP_FSMODE, FsMode);    % sampling rate control register (mode value)
PutDSPProperty(hBoard, DSP_FSVALUE, FsActual); % actual sampling rate (in Hz)

% Note:  uncomment the DSP_FRMSIZ property below to cause UserProc() function to be inserted into
% the real-time data stream processing on the DSP board.  This file is included in the "DSP
% Source Code Interface" software, and is intended to serve as an “insertion point” for user-
% defined real-time C (or .asm) code which runs on the DSP board.  This approach enables the
% built-in analog I/O interrupt and buffer processing  on the board to run normally, initialized
% from MATLAB, and also keeps  all host interface and communication routines intact.  This allows
% a focus only on the application-specific, C programming needed for real-time operation, and
% minimizes the amount of programming

PutDSPProperty(hBoard, DSP_FRMSIZ, BufSize);


% create data buffer; note that data must be scaled to be reasonable values
% for D/A converter(s)

for (n=1:BufSize)     % output buffer is 1 Hz sine wave

   buf1(n) = 10000*sin(2*pi*n*1000/FsActual);
   buf2(n) = 10000*sin(2*pi*n*1000/FsActual);

end

% pre-load DSP board memory with data buffers; both buffers must be preloaded for
% continuous operation

% note that download depends on whether board has linear or Harvard memory architecture

if (MemArch == DS_GMA_VECTOR)   % board has vector memory architecture

    Status = PutMem(hBoard, DS_GM_VECTOR_DATA_X, OutBaseAddr, DS_GM_SIZE16_CVT, buf1, BufSize);
    Status = PutMem(hBoard, DS_GM_VECTOR_DATA_Y, OutBaseAddr, DS_GM_SIZE16_CVT, buf2, BufSize);

else  % board has linear or Harvard memory architecture

    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr, DS_GM_SIZE16_CVT, buf1, BufSize);
    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr+BufSize, DS_GM_SIZE16_CVT, buf2,
BufSize);
end

if (Status == NULL)
    errstr = ['problem with PutMem; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);
end


% *** Example application specific code
% note:  perform all GetSymbolAddress() calls prior to running code when possible

my_var = 0;
```

 Copyright © Signalogic 2003-2005

```
my_var_addr = GetSymbolAddress(hBoard, NULL, '_my_var');
dec2hex(my_var_addr)

% ***

% Start board and wait for buffer to be output.  Note above that we filled
% second buffer also, to avoid "screech"--because DSP will start output
% on second buffer before we can stop the processor

% Note:  first buffer will be output as soon as board starts running; for
% continuous % output, "Buffer Ready" callback messages must be processed,
% and the RegMsgWnd() and WaitForBuffer() functions must be used

RunProcessor(hBoard, 1);  % run; processor #1 only

fCont = 10;               % we will output 10 buffer of samples

while fCont > 0,

  % wait to make sure first buffer has been output
  WaitForBuffer(hBoard, BufNum, NULL, DS_WFB_POLLED + DS_WFB_SYNC);

% *** application-specific example code

  % monitor application symbols...

  GetMem(hBoard, DS_GM_LINEAR_DATA_RT, my_var_addr, DS_GM_SIZE32_LONG_CVT, my_var, 1);

% ****

  if (MemArch == DS_GMA_VECTOR)  % board has vector memory architecture

    if (BufNum == 0)
        Status = GetMem(hBoard, DS_GM_VECTOR_DATA_X, TimBaseAddr, DS_GM_SIZE16_CVT, buf_plot,
BufSize);
    else
        Status = GetMem(hBoard, DS_GM_VECTOR_DATA_Y, TimBaseAddr, DS_GM_SIZE16_CVT, buf_plot,
BufSize);
    end
  else  % board has linear or Harvard memory architecture

    Status = GetMem(hBoard, DS_GM_LINEAR_DATA_RT, TimBaseAddr+BufNum*BufSize, DS_GM_SIZE16_CVT,
buf_plot, BufSize);
  end

  buf = buf_plot;  % store acquired data in variable buf for PutMem use

  if (Status == NULL)
    errstr = ['problem with GetMem; error code = ' num2str(GetEngErrStat(hEngine))];
    disp(errstr);
  end

% If you want to change the output as you are acquiring data, uncomment the
% following session which feed the input to the output directly
%
%  if (MemArch == DS_GMA_VECTOR)  % board has vector memory architecture
%
%    if (BufNum == 0)
%       Status = PutMem(hBoard, DS_GM_VECTOR_DATA_X, OutBaseAddr, DS_GM_SIZE16_CVT, buf,
BufSize);
%    else
%       Status = PutMem(hBoard, DS_GM_VECTOR_DATA_Y, OutBaseAddr, DS_GM_SIZE16_CVT, buf,
BufSize);
%    end
%   else  % board has linear or Harvard memory architecture
%
%    Status = PutMem(hBoard, DS_GM_LINEAR_DATA_RT, OutBaseAddr+BufNum*BufSize, DS_GM_SIZE16_CVT,
buf, BufSize);
%  end
%
%  if (Status == NULL)
```

 Copyright © Signalogic 2003-2005

```
%    errstr = ['problem with GetMem; error code = ' num2str(GetEngErrStat(hEngine))];
%    disp(errstr);
%  end

  BufNum = 1 - BufNum;  % toggle buffer number

  fCont = fCont -1;     % if you want the loop to go forever, don't decrement fCont

end

ymax=max(buf_plot);          % set y-axis maximun
ymin=min(buf_plot);          % set y-axis minimum

figure(gcf);                 % use current figure, if one

axis([0, BufSize-1, ymin, ymax]);
pt = min(BufSize, 1000);
plot(buf_plot(1:pt));        % show part of the data

% done; leave board in disabled state, clean up handles, engines
if (fBoardInitialized == TRUE)
    DisableBoard(hBoard);
end

if (hBoard ~= NULL)
    FreeBoard(hBoard);
end

if (hEngine ~= NULL)
    EngineClose(hEngine);
end
```

5.3 Building Linux Software

To build the software modules listed in the subsections below, you will need to have root access. There are three (3) options for gaining root access:

1. Log-in as root user:

```
Login: root
password: veyung
```

2. Acquire sudo access rights to commands ldconfig, depmod, rmmod, and insmod. To gain sudo access rights, type:

```
sudo  <command>
```

followed by the command. Then enter your user password.

3. Switch user to root by typing

```
su
```

at the command prompt and enter the root password.

Notes

1) Under /usr/local/share/, the subdirectories HwMgr, EngMgr, SigFModule have attributes user nobody and group nobody.

2) On low-power consumption systems such as a Lippert CoolRunner II, a full make of the e1 build may take up to 5 minutes because the processor speed is relatively slow, about 300 MHz.

3) The methods above discuss root access. For information purposes, note that on the Lippert system, a normal user account has also been created with username temujin and password veyung.

5.3.1 Rebuilding boardTest and dataScope Test Programs

To rebuild BoardTest test program, perform the steps shown in the screen capture below:

5.3.2 Rebuilding the E1-Audio Test Program

To build the Signalogic E1-Audio test program from the main extraction folder (see section 2.2.3 above, Directory Structure), change to the HwMgr folder and execute the make command, as shown below:

```
shell/> cd HwMgr
```

```
shell/HwMgr> make e1
```

This will create a new e1test.o executable module, which is the E1-Audio version of the XDScope test program that comes with the Linux version of DirectDSP software. To run the E1-Audio test program, see section 3.5.3.1 above, Running the E1-Audio Test Program. Note that it is not necessary to first gain root access to rebuild e1test, since it is an application-level program.

5.3.3 Rebuilding the DUART Test Program

To build the Signalogic DUART test program from the main extraction folder (see section 2.2.3 above, Directory Structure), change to the Comm-Prog folder and execute the make command, as shown below:

```
shell/> cd Comm-Prog

shell/HwMgr> make CommProg
```

This will create a new CommProg.o executable module. To run the CommProg test program, see section 3.5.3.2 above, Running the DUART Test Program. Note that it is not necessary to first gain root access to rebuild CommProg, since it is an application-level program.

5.3.4 Rebuilding LKM Drivers

This section describes how to rebuild LKM (loadable kernel module) drivers used by SigC5xxx and SigC6xxx boards under Linux.

Note that LKM drivers always be rebuilt upon first installing hardware or re-building the kernel, to avoid incompatibilities or mismatches that could affect the interface between software components.

5.3.4.1 Rebuilding SigC5xxx and SigC6xxx PC104 LKM Drivers

To rebuild SigC5xxx and SigC6xxx PC104 board LKM driver, follow the steps below.

1) First, log-in as root user (see section 5.3 above, Building Linux Software). Then remove the driver from memory, by entering:

```
shell/> /sbin/rmmod SigFM
```

2) Next, from the main extraction folder (see section 2.2.3 above, Directory Structure), change to the SigFModule folder and execute the make command:

```
shell/> cd SigFModule

shell/SigFModule> make all
```

3) Last, re-install the LKM driver, as shown below

```
shell/SigFModule> insmod SigFM.o
```

NOTE: *In order for the "make all" command to work correctly, do not attempt to install the loadable kernel module driver or run individual pieces of the makefile separately without knowing EXACTLY what you are doing.*

5.3.4.2 Rebuilding the SigC641x and SigC5561 PTMC LKM Drivers

Following are preliminary instructions for rebuilding the SigC641x and SigC5561 PTMC card LKM driver.

1) First, install the source code into a known location and change to that directory.

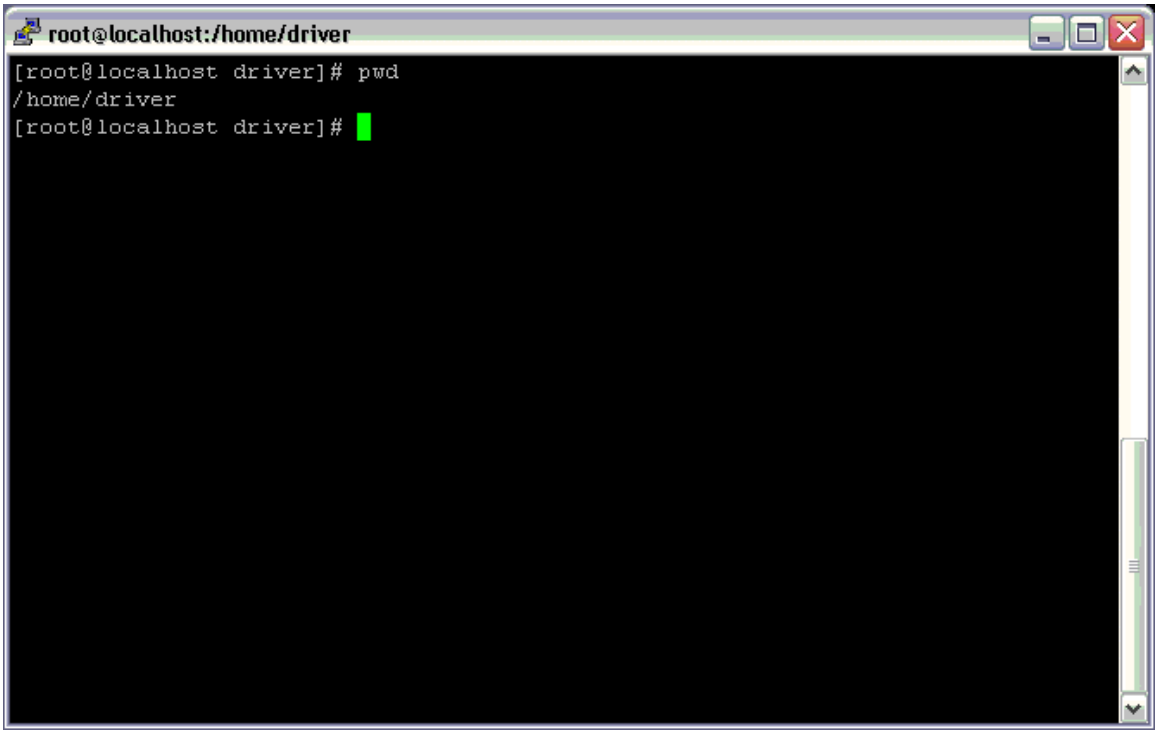


Figure 5-8, SigC641x and SigC5561 PTMC Card Driver Source Location

2) Build the driver. The kernel source code must be installed for the module to compile correctly and the makefile looks for the necessary headers files with the gcc switch

```
-isystem /lib/modules/`uname -r`/build/include
```

After executing the above command, you should see a display similar to Figure 5-9 below:

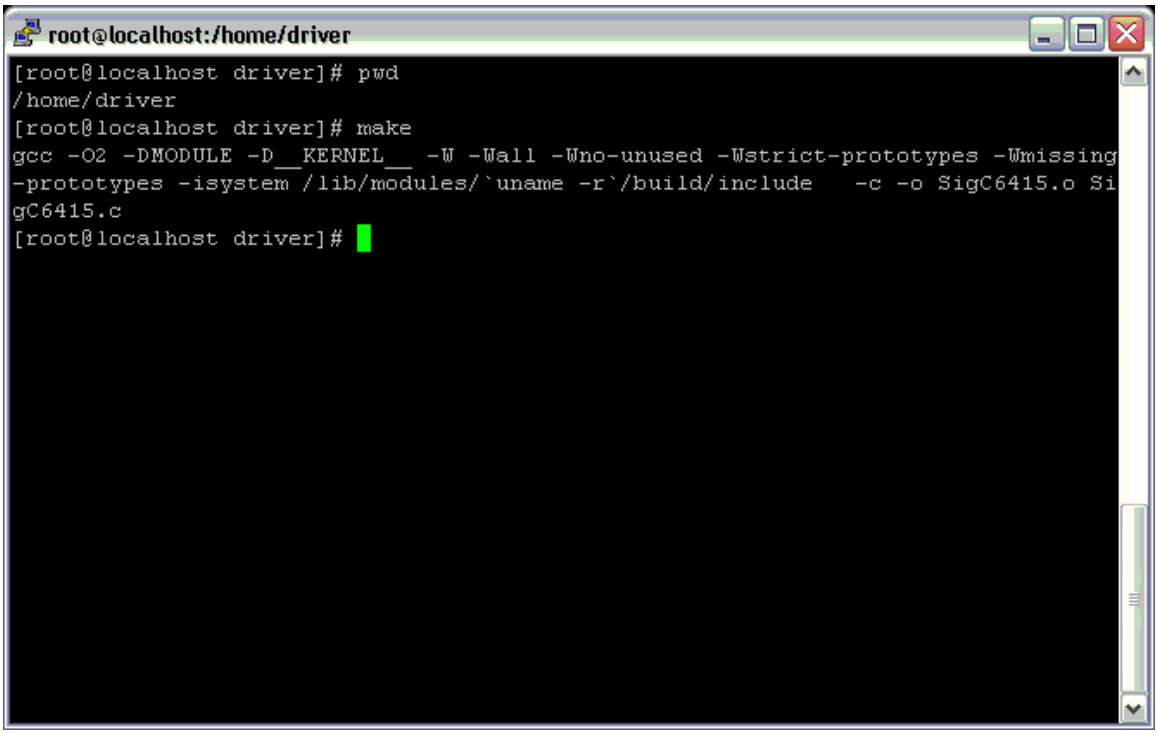


Figure 5-9, SigC641x and SigC5561 PTMC Card Driver Source Build

5.3.5 Application Layer Source Code

Below is a hierarchy chart explaining relationship between application layer source code and DirectDSP shared object libraries and drivers.

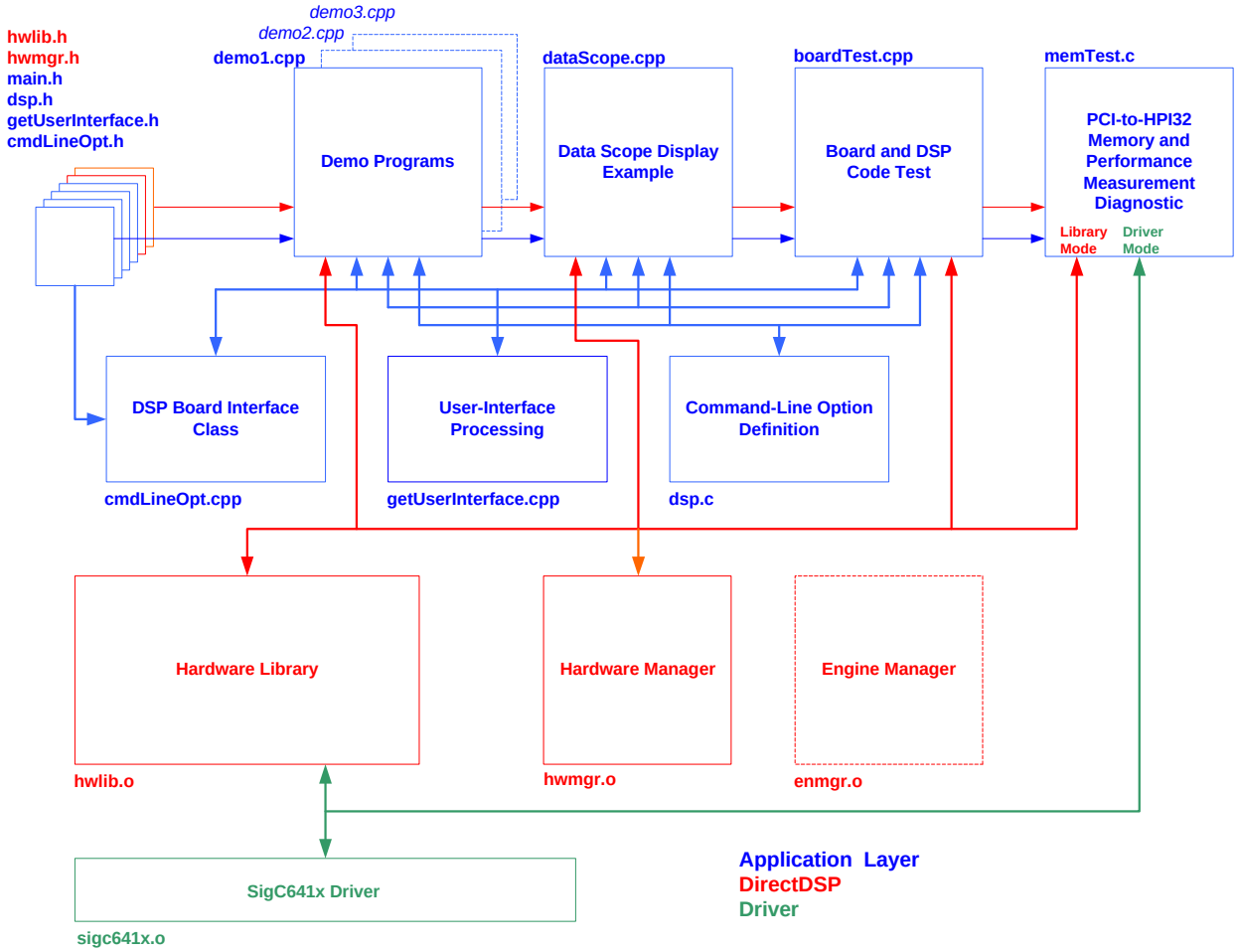


Figure 5-10, Application Source Code and DirectDSP Relationship

Below are source code excerpts intended to demonstrate a typical Linux C and C++ code application using DirectDSP to communicate with SigC5xxx or SigC6xxx boards. The source code excerpts are related to each other as shown in the above hierarchy chart.

The program used for illustration purposes is boardTest, a generic, basic level C/C++ program that performs the following functions:

- Assigns a board handle
- Initializes board and driver; checks the board to make sure it's functioning correctly
- Downloads a DSP executable file (COFF file) generated by Code Composer Studio IDE from Texas Instruments
- Initializes variables and buffers in the DSP code and prepares it to run
- Runs DSP code
  Communicates with the DSP code during execution to determine when to transfer data buffers to and DSP memory for processing, and when to upload data buffers when ready

 Copyright © Signalogic 2003-2005

Source code excerpts from the boardTest program included here are summarized in the table below:

Filename	Description	Comments
boardTest.cpp	Main example program	Demonstrates DirectDSP APIs
dsp.cpp	DSP board class	Simplified, streamlined C++ interface to DirectDSP
getUserInterface.cpp	User-interface processing	Reads command-line options (for text based demo) or displays Hardware Manager dialog (for GUI based demo)
cmdLineOpt.cpp	Defines and displays command-line options	
hwlib.h	DirectDSP include file for Hardware Library shared object	
hwmgr.h	DirectDSP include file for Hardware Manager shared object	
main.h	Include file used for all test programs	
dsp.h	DSP board class include file	
getUserInterface.h	User-interface include file	
cmdLineOpt.h	Command-line definition include file	

Figure 5-11, Source Code Excerpts from boardTest Example Program

 Copyright © Signalogic 2003-2005

6 Developing and Debugging DSP Code

This section discusses methods of DSP code development, including using Code Composer Studio (CCS) software from Texas Instruments, using DirectDSP software from Signalogic, and combinations of the two methods.

Signalogic supports both "CCS centric" and non-CCS methods of development and debug. There are advantages and disadvantages of each approach, and sometimes a complimentary mix is most effective. Also, we recognize that choices for development can debug configuration of C5xxx and C6xxx hardware – and which software tools to use -- can be largely a matter of individual engineer preference.

Note that ***in all cases, CCS is recommended to maintain and build DSP projects***, including compilation, assembly, linking, and configuration of DSP/BIOS real-time operating system and CSL (Chip Support Library).

CCS Debug Philosophy

Some engineers prefer to use only CCS tools for debug, performing all code download (to DSP hardware) and debug via JTAG emulation. CCS is a mature, robust environment that allows DSP code to be debugged at a very low level, including single-step, breakpoint, memory display and disassembly, and chip register view and monitoring.

The Texas Instruments CCS tools are extremely good, and once you get used to them, that comfort and efficiency can mean a great deal in the lab. Signalogic's development approach and philosophy very much takes into account the benefit of this efficiency; at any given moment you can walk through a Signalogic lab and see engineers using CCS tools for both C5xxx and C6xxx project development and debug.

More information on using CCS (Code Composer Studio) software can be found in section 4 above, Building DSP Application Code.

DirectDSP Debug Philosophy

Other engineers prefer to avoid JTAG when possible because it can be slow and tedious, and rely on host software tools to download executable DSP code and "see" algorithm results as the code is running in real-time.

The main reasons why Signalogic offers an alternative method to CCS involve real-time and performance issues. In some situations, CCS can be more hindrance than help, for example when there is a need to see I/O data in real-time without pausing or re-starting the processor, or a need to rapidly move bulk data to/from host GUI or application level software. Some examples in the latter category include:

- Data logging
- Real-time measurement and analysis

 Copyright © Signalogic 2003-2005

- Bit-exactness comparison purposes
- When program and data downloads and transfers start to take a long time and impact development time in the lab

For host GUI and application software interface, the DirectDSP software can be used. As opposed to JTAG, which is a serial interface (from 10 to 50 MHz clock, depending on emulator type), DirectDSP uses a bus-interface to communicate directly with the HPI port on the DSP(s): ISA bus for PC/104 cards, PCI bus PCI cards, parallel port for TI DSK boards, etc.

Customer experience shows the most common reason to not use CCS+JTAG -- or to keep it in play but also enable a simultaneous bus-interface -- is when a host-side GUI is a key part of the system, and either a) there is a compelling reason (such as a chunk of existing code) to use popular development environments such as Visual C/C++, VB, or MATLAB, or b) the GUI environment does not matter, but RTDX does not allow screen items and data displays to be updated quickly enough.

For more information on DirectDSP software, how it can be used, and how it works, see section 5 above, Building Host Application Software.

How to Start a Project

A typical starting point for a new project is CCS running on a WinXP system. If the application then requires Visual C++ or MATLAB accesses to the DSP and hardware, the next step is to add that to the mix. In many situations there is merit to simulating floating-point algorithms in MATLAB and then turning off .m code sections and turning on DSP C code sections. It is completely possible to use MATLAB flags to enable DSP code section-by-section to ensure – in a step-by-step, controlled manner – that simulation results match real-time results. Using this approach, a point can be reached where all real-time code is running on the DSP and the GUI is implemented in Visual C/C++, Visual Basic, or MATLAB.

For debug functionality, there is little overlap between the CCS IDE and host software GUI examples offered by DirectDSP. DirectDSP software does not currently provide single-step, breakpoint, register view, or memory display functions. Instead the examples provided by Signalogic focus on accessing variables and buffers in user-defined application DSP code, for instance using DSGetSymbolAddr() to get the physical location from a symbol name, and then using memory transfers with the physical address to "see" (monitor) these symbols as they update in real-time.

6.1 Host Mode and UserProc Debugging

Below are some examples of Host Mode Operation.

6.1.1 Hypersignal-Macro Simple FFT Function

1. If needed, on the CCS main menu bar, select Project | Build (or F7) to build the project.
2. Note that as the last step of the build, the resulting executable binary file (.out file in COFF format) will be copied to the \hsmacro subdirectory. Note that the .out, .map, and .obj files will be located in the same directory as the .pjt files. For a list of default .out filenames, see the table in section 4.1.7 above, CCS Build Output, File Types, and File Locations.

1. Run Hypersignal-Macro as follows:

   Start button on Task Bar
   select Programs
   select Signalogic DSP Software | Hypersignal – Macro

2. Go to the System Config Menu as in Figures 6-1 and 6-2 below. Enter sigc54xx for C54xx processors and sigc67xx or sigc62xx for C6xxx processors.

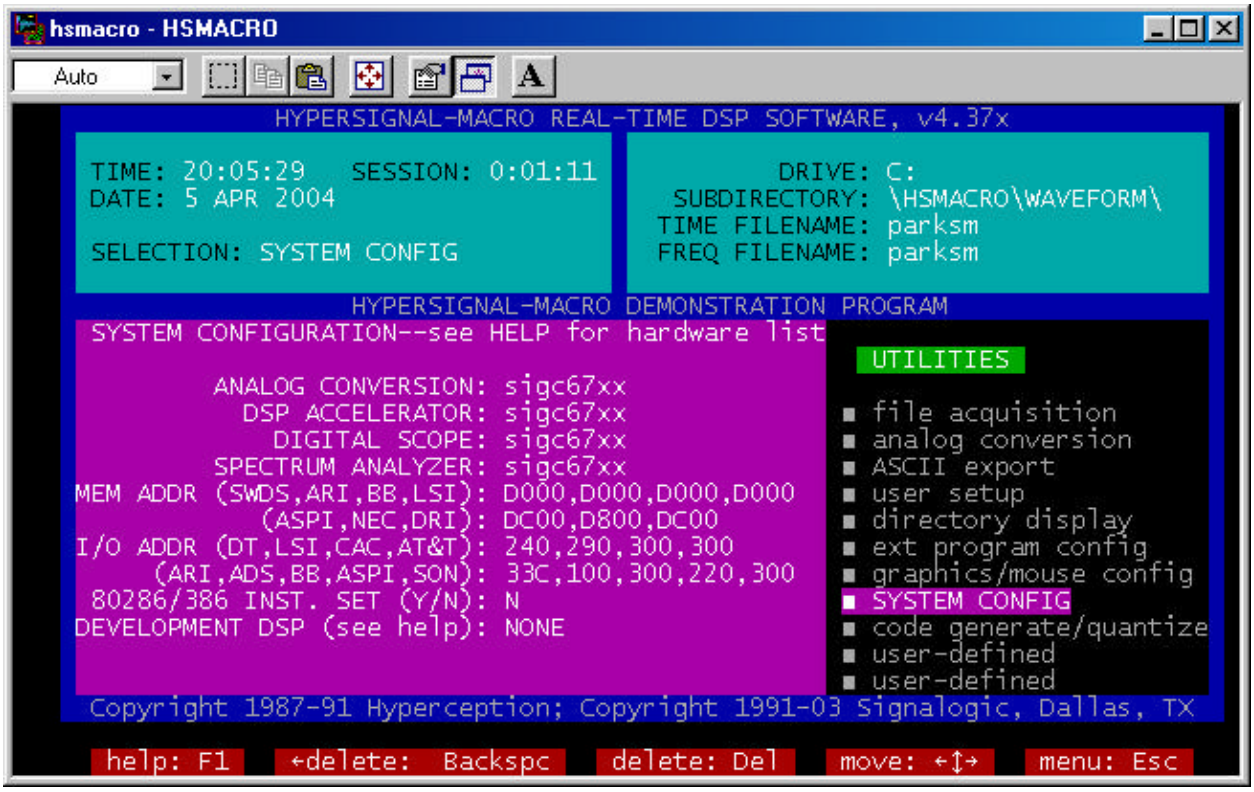


Figure 6-1, Hypersignal-Macro System Configuration, SigC67xx Hardware Selected

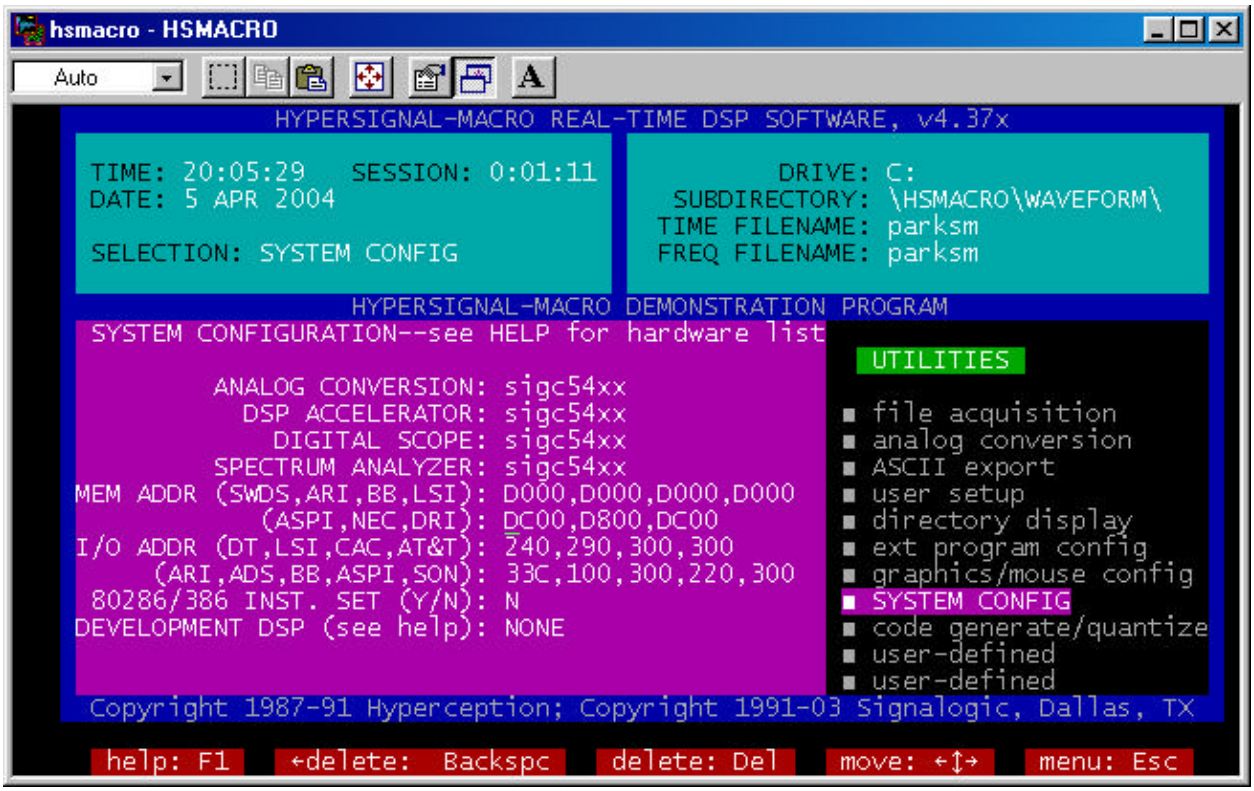


Figure 6-2, Hypersignal-Macro System Configuration, SigC54xx Hardware Selected

3. Press ESC key to return to the main menu,, go to Time Domain menu (at left on main menu), then select FFT Generation menu item.
4. Types in a waveform file name such as parksm, as shown in Figure 6-3 below.

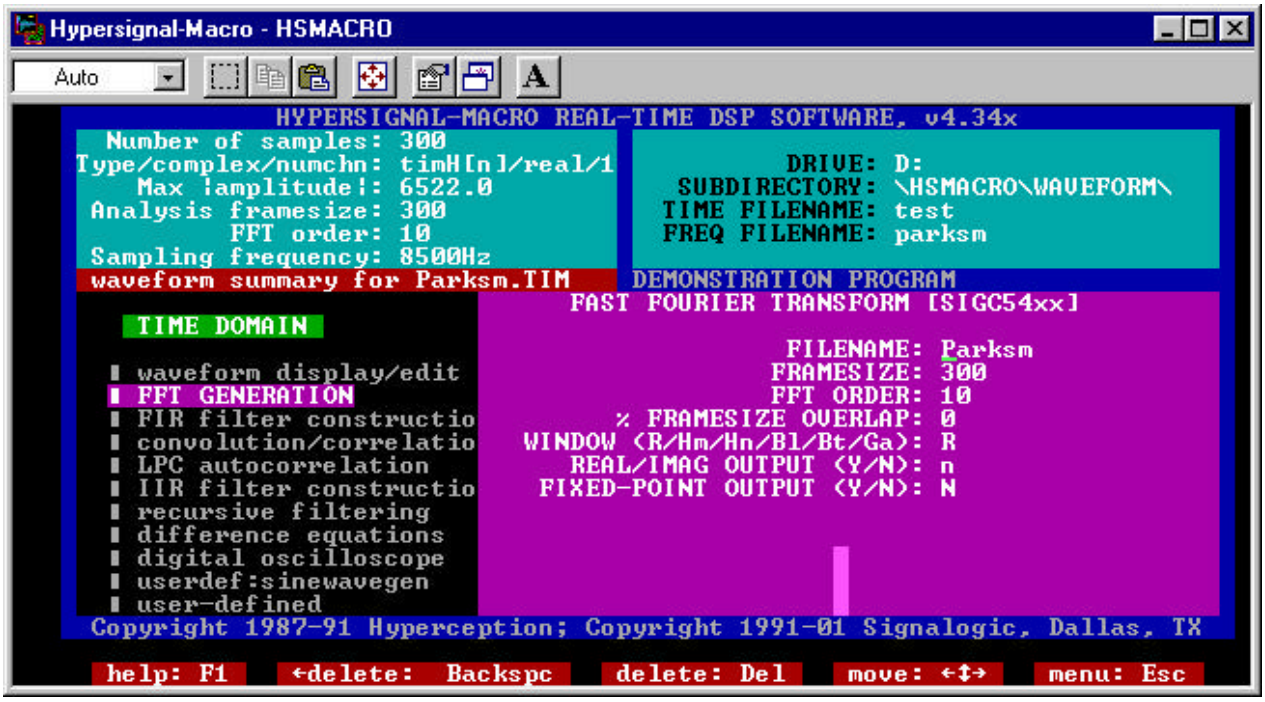


Figure 6-3, Hypersignal-Macro FFT Generation

5. Press Ctrl+Enter to run the FFT. You can change the Framesize, FFT order, Window type, and other parameters if desired.
6. If you see a error message comes say invalid interface card address as Figure 6-4 below, you should exit out of Hypersignal-Macro by hitting the following key sequence: ESC, q (quit), and a (abort) or e (exit and save).

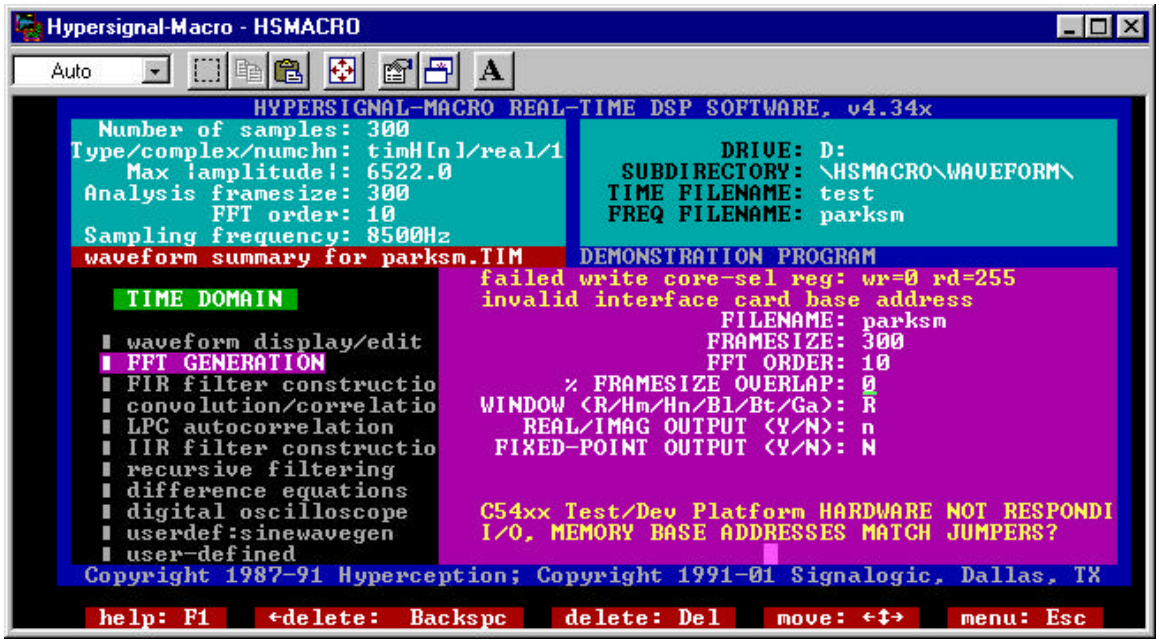


Figure 6-4, Hypersignal-Macro Error Message

7. An incorrect setting in the Hypersignal Configuration for the base address could cause this error. To fix that, do the following:
    a. Open hsmacro.cnf in \hsmacro directory in Notepad.
    b. Go to the 5th number from the bottom of the file.
    c. If your have PC/104 card, your base-address should be 0x320; therefore the 5th number from the bottom should be 800.
    d. If your board base address set to 0x340, the 5th number from the bottom should be 832.
    e. Modify the number accordingly, save and exit the file.
    f. Go back to step 7.
8. If you run FFT and see following message as in Figure 6-5 below, press ‘Y’ key.

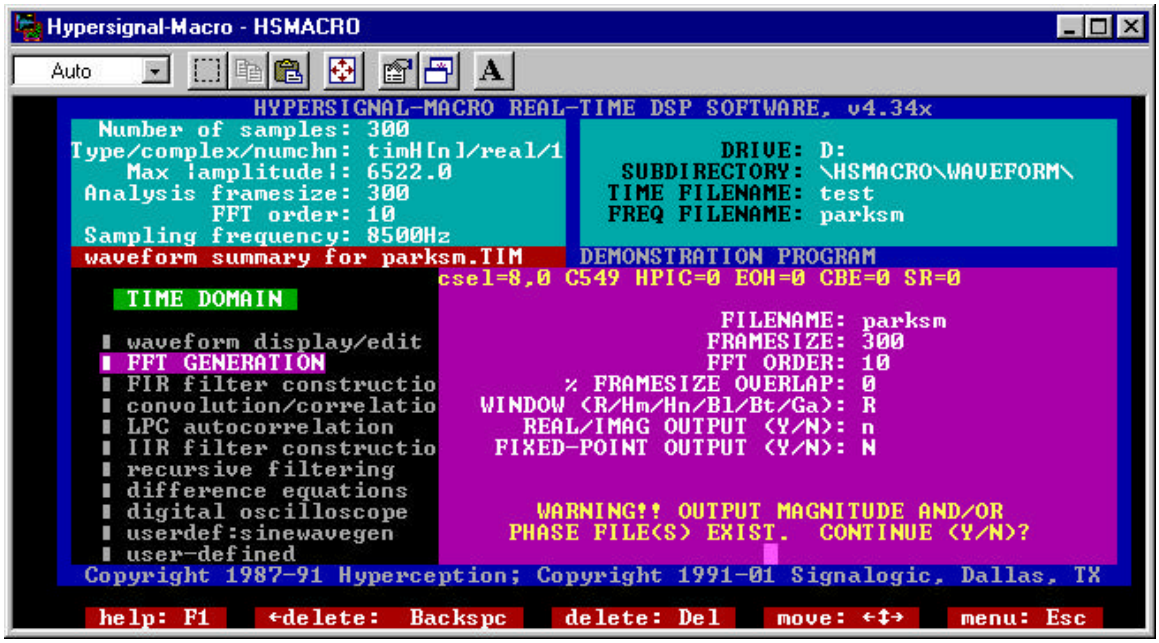


Figure 6-5, Run Hypersignal-Macro FFT Generation

9. Go to Magnitude Display under Frequency Domain Menu as in Figure 6-6 below. Press Ctrl+Enter.

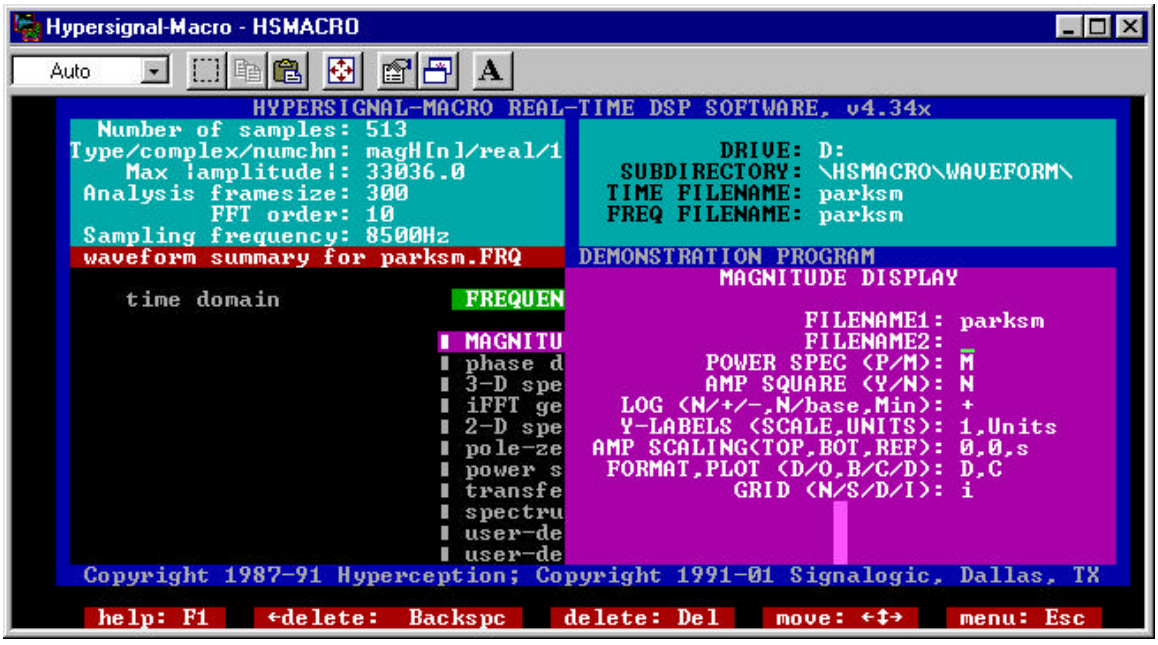


Figure 6-6, Hypersignal-Macro Magnitude Display

10. You should see the answer displayed as in Figure 6-7 below.

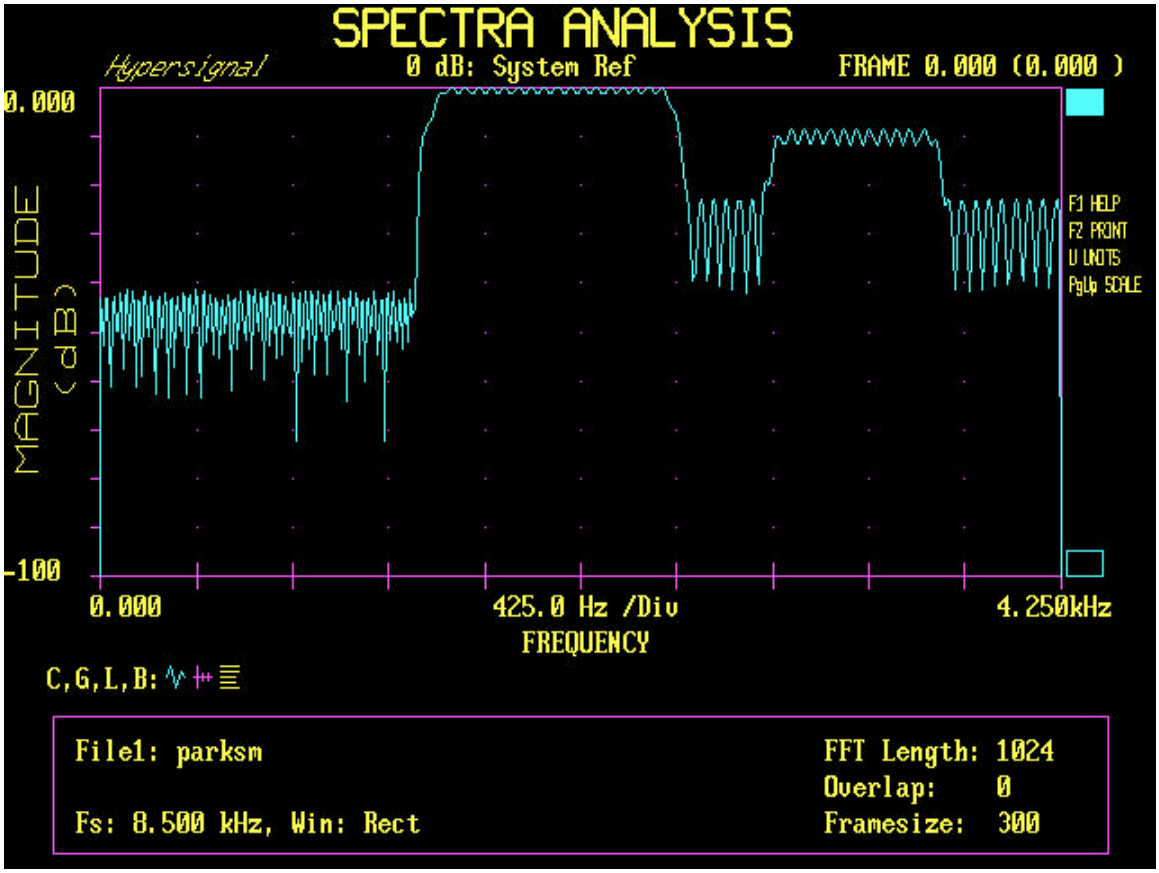


Figure 6-7, Magnitude Display for FFT Generation of "Parksm" Waveform

 Copyright © Signalogic 2003-2005

6.1.2 UserProc: Adding C Code to Real-Time DSP Processing

All C5xxx and C6xxx SCI and DSP/BIOS builds include the file userproc.c which contains a function called UserProc(), which serves as a general-purpose, “getting-started” entry-point for user-defined real-time C code.

UserProc() essentially inserts user-defined C code into the DSP code processing the real-time analog or digital data stream. There are four (4) important ideas underlying the UserProc() method:

- Hide details of the DSP chip and DSP board--especially analog I/O, which tends to vary widely from board to board and can be complex to deal with from a software point of view
- Allow user-defined 'C' code to process analog input/output data on a buffer, or "frame" basis, and not deal with individual samples
- Allow real-time recording of intermediate results to waveform file or memory for debugging purposes
- Make the DSP code modification cycle, or "turn-around time", simpler and faster

UserProc() is called after each analog input/output buffer combination is processed by the analog I/O drivers and has been presented to the foreground processing layer in the DSP operating system. This occurs before analog input buffers are sent to host PC software, such as Hypersignal, or Visual C/C++, Visual Basic, or MATLAB program using DirectDSP software. In this way, both analog input buffers (before they reach host application-level software) and output buffers (after they are sent by host application-level software) can be modified by real-time DSP code.

This type of operation is known as "frame-based real-time processing", where background interrupt service routines use automatic serial port DMA onchip circuitry to assemble analog input samples into a series of frames, or buffers, which are then processed to the foreground algorithm for processing on a "frame" basis.

Data flow between analog I/O and UserProc() is shown in Figure 6-8 below.

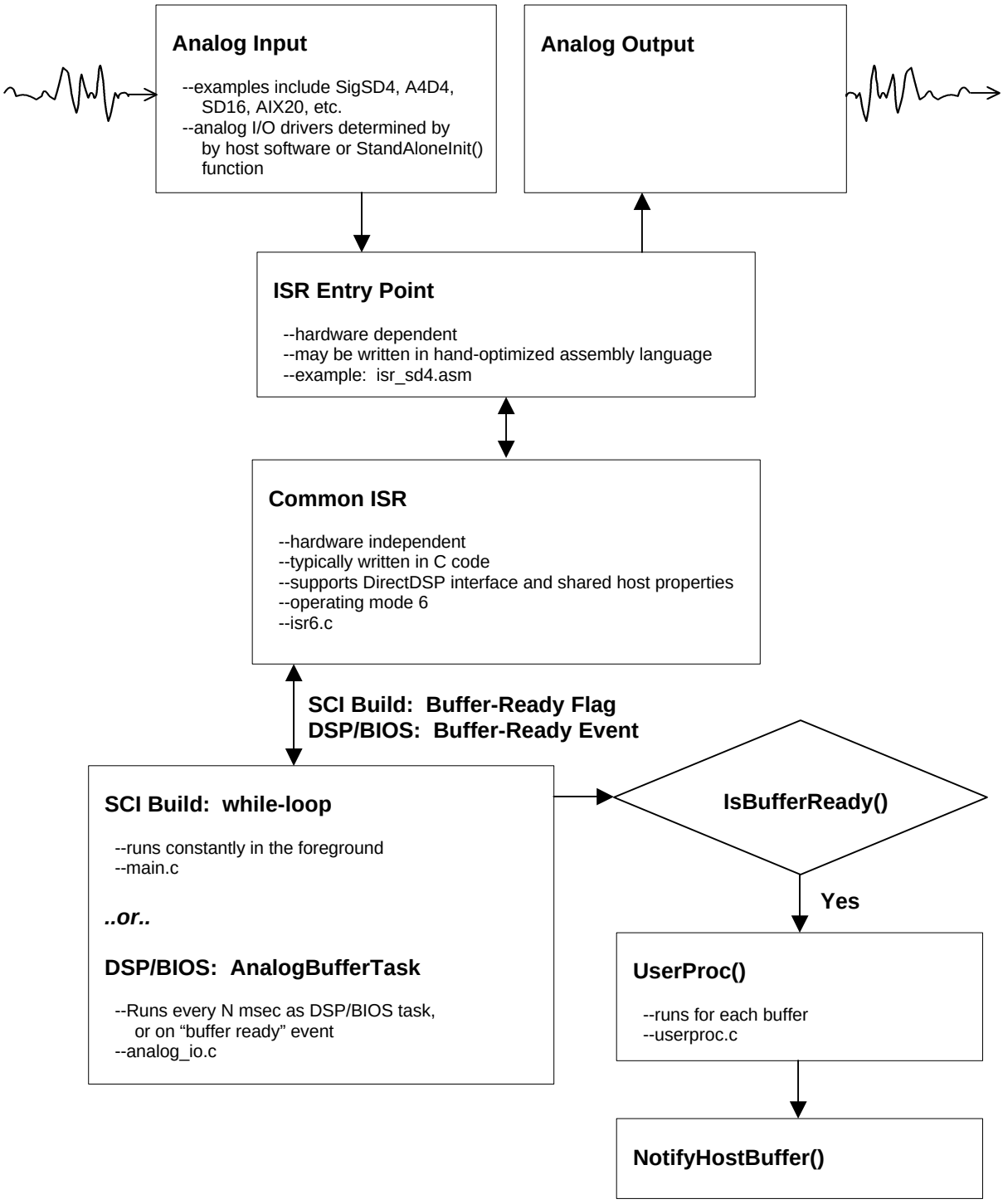


Figure 6-8, Real-Time I/O Data Flow

Note that analog I/O drivers used by DSP code are determined either by

 Copyright © Signalogic 2003-2005

- In Host Mode by host software (Hypersignal, MATLAB, Visual C/C++, Visual Basic). In Hypersignal software this is done using the System Config menu selection; in DirectDSP-enabled software by the Hardware Manager dialog

- In Stand-Alone Mode by StandAloneInit() function called by DSP boot code

For SCI builds operating in default mode 6, UserProc is called from main() as shown in the C source code below:

```
void main() {

   fHardwareOk = InitBoard(RTC_IB_DUART | RTC_IB_DIGITAL_IO | RTC_IB_LED | RTC_IB_ANALOG_IO);

/* for SCI project, enter indefinite real-time loop and start processing buffers immediately...
*/

   while (TRUE) {

      AnalogBufferTask();
   }

}

void AnalogBufferTask() {

/* call to InitBoard() here only relevant to DSP/BIOS build; see comments in main() above */

   if ((fHardwareOk & RTC_IB_ANALOG_IO) == 0) fHardwareOk |= InitBoard(RTC_IB_ANALOG_IO);


   if (IsAnalogBufferReady()) {

   /* process next analog I/O buffer */

      ProcessAnalogBuffer();

   /* notify host that a buffer is ready */

      NotifyHostBuffer();
   }
}
```

For DSP/BIOS builds operating in default mode 6, UserProc() is called from AnalogBufferTask(),when a new analog I/O buffer is available, as shown in C code for main() below, and AnalogBufferTask() above.

```
void main() {

/* For DSP/BIOS project, ok to initialize all hardware except for analog I/O; wait
   until all tasks start to do that.  This is required because at this point -- inside
   main() -- DSP/BIOS is not done with interrupt and other sensitive hardware and chip
   initialization.

   Note there is one additional call to InitBoard() in AnalogBufferTask(), below.
   AnalogBufferTask is activated either by DSP/BIOS at regular intervals or due to a
   Buffer-Ready event. */

   fHardwareOk = InitBoard(RTC_IB_DUART | RTC_IB_DIGITAL_IO | RTC_IB_LED);

/* for DSP/BIOS project, wait for task start to begin processing buffers... */
```

```
}
```

The default period for AnalogBufferTask() activation by DSP/BIOS is 10 msec, but should be adjusted depending on sampling rate and buffer size. A recommended rule-of-thumb for calculating the period is to use 5% to 10% of the time represented by each buffer, which can be calculated as:

$$\frac{\text{Buffer Size (samples)}}{\text{Sampling Rate (Hz)}}$$

For example, if the buffer size is 256 samples and sampling rate is 16 kHz, then AnalogBufferTask() might run every 1 to 2 msec.

Parameters passed to UserProc() include pointers to the just-acquired input buffer and next output buffer, length of each buffer in “sample groups”, and number of traces (channels) contained in each sample group. Note that a sample group is more than one sample if number of traces is > 1. For example, a three-channel acquisition would cause buffers to contain interleaved data in the form ch0, ch1, ch2, ch0, ch1, ch2, etc. In that case, the actual total length of the buffer is nLen*nNumTraces

Below is the default source code listing for the basic userproc.c routine, which is configured for simple loopback operation (loop input back to output) before user-defined modification:

Default UserProc Routine

```
void UserProc(void* ptrIn, void* ptrOut, int nLen, int nNumTrace) {

#if defined(C5xxx) && defined (_LONG_AD_CONVERTER_)

  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)

#else  /* C5xxx 16-bit or C6xxx N-bit */

  #define x ((int*)ptrIn)
  #define y ((int*)ptrOut)

#endif

short int n;

   for (n=0; n<nLen; n++) {

      for (m=0; m<nNumTrace; m++) {

         y[n*nNumTrace + m] = x[nNumTrace*n + m];  /* loop input back to output */

      }
   }
}
```

Some important notes about the above code structure:

1) Each input buffer ("x") contains A/D values represented as right-justified, N-bit signed integer values, where N is the natural word-length of the DSP device. For C5xxx boards N == 16, for C6xxx boards N == 32. An exception occurs for C5xxx boards when A/D converter resolution exceeds 16-bits, in which case A/D values must be represented by 32-bit integers. Note that for C6xxx boards, 32-bit samples are used regardless of A/D converter resolution.

2) Each output buffer ("y") contains 16-bit D/A values, as described in 1) above.

3) The nNumTrace parameter usually corresponds to the number of channels. However, it is more accurate to say "traces" instead of "channels" because the Channel List property specified in Hypersignal or DirectDSP-enabled software may actually be an arbitrary ordering of channels, in which case, for example, channel 0 may not correspond to trace 0.

If the nNumTrace parameter is > 1, then data is stored as interleaved values, in repeating groups of nNumTrace values. For example,

```
tr0,tr1,tr2,tr0,tr1,tr2, ...
```

for 3-trace acquisition or signal output.

4) **NOTE, VERY IMPORTANT**: *UserProc() must complete processing before the next analog I/O buffer is ready!* Otherwise, real-time operation is lost. This is stating the obvious, but nevertheless it pays to calculate buffer time vs. processing time carefully and accurately and understand the amount of real-time overhead available.

Application Examples

A few simple application examples are given below.

To rebuild the project after code has been modified as shown in the examples below -- or otherwise as needed – go to the CCS main menu bar and select Project | Build (or press F7). For complete instructions on re-building projects, see section 4.1 above, CCS Project Build and Run Sequence.

Signal Arithmetic Example

A very simple example of user-defined real-time 'C' code is basic signal arithmetic. In this example, the DSP algorithm scales and adds a constant to input channel 0 data, and also loops the data back to the output. The difference equation for this example could look something like the code shown below.

```
y[n] = 0.75*x[n] + 1000;
```

First, userproc.c is modified to look like:

 Copyright © Signalogic 2003-2005

```
void UserProc(void* ptrIn, void* ptrOut, int nLen, int nNumTrace) {

#if defined(C5xxx) && defined (_LONG_AD_CONVERTER_)

  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)

#else  /* C5xxx 16-bit or C6xxx N-bit */

  #define x ((int*)ptrIn)
  #define y ((int*)ptrOut)

#endif

short int n;

   for (n=0; n<nLen; n++) {

      for (m=0; m<nNumTrace; m++) {

         y[n*nNumTrace + m] = 0.75*x[n*nNumTrace + m] + 1000;

      }
   }
}
```

Running Sum Example

A slightly more complex example is to combine basic signal arithmetic with recursive input (i.e. input from previous output samples). In this example, the DSP algorithm performs a running sum on input channel 0 data, with no loopback to the output. The difference equation for this example could look something like:

```
    y[n] = y[n-1] + (x[n] - x[n-M])/M
```

where M is the length of the running sum, in samples. First, UserProc() is modified to look something the code shown below.

```
#define N 4096  /* define N as maximum buffer size */
#define M 32    /* define M as length of running sum */

short int ySum[N];    /* create storage of one buffer, assume it is initialized to zero */

short int xPrev[M];   /* create storage of length of running sum, to hold samples at end of each
                         previous buffer ("memory" between successive frames) */

void UserProc(void* ptrIn, void* ptrOut, long nLen, short int nNumTrace) {

#if defined(C5xxx) && defined (_LONG_AD_CONVERTER_)

  #define x ((long*)ptrIn)
  #define y ((long*)ptrOut)

#else  /* C5xxx 16-bit or C6xxx N-bit */

  #define x ((int*)ptrIn)
  #define y ((int*)ptrOut)

#endif

short int n,xn;
long static lSum = 0;
```

```
   for (n=0; n<nLen; n++) {

      if (n - M < 0) xn = xPrev[n];  /* if index is before start of current buffer, save in
                                        static buffer for use with next frame */
      else xn = x[n-M];

      lSum += x[n] - xn;  /* calculate x term sum */

      ySum[n] += (short int)(lSum/M);  /* calculate running sum */
   }

   for (n=0; n<M; n++) xPrev[n] = x[nLen-n];  /* store last M points for use by next frame */

    for (n=0; n<nLen; n++) x[n] = ySum[n];  /* store processed data in input buffer */
}
```

Notes

1) In the running sum example above, nNumTrace is assumed to be 1 to make it more readable.

2) The use of a long division intermediate calculation above (lSum/M) is not suggested for real-time processing; it is used only to make the example general and portable across different DSP devices. A better approach would be to make M a power of 2, so the division could be achieved by shifting the lSum term right.

6.1.3 Probing Real-Time C Code from Hypersignal

To activate UserProc in Hypersignal operation, enter:

```
'a,rt,r'
```

or

```
'a,rtd,r'
```

in the ACQ/GEN field in the Analog Conversion function. The above examples are special "coded entries" that Hypersignal interprets as UserProc-related operating modes, rather than output or stimulus wave filenames, as would normally be the case. These entries function as follows:

- 'rt' entry activates UserProc() and enables real-time simultaneous input/output operation
- 'rtd' entry works the same way, but in addition enables continuous recording to waveform file of real-time input processed by Userproc() DSP code

Note that maximum sampling rates at which continuous operation (no gaps or data discontinuity) can be achieved are lower when file recording is enabled. In all cases, maximum sampling rates at which real-time operation can be maintained are heavily system dependent. Important factors include DSP chip clock rate, DSP external memory access time, and efficiency of UserProc C code. In the case of waveform file recording, factors also include host PC speed, disk drive speed and controller type, ISA or PCI bus speed, etc.

Note that if during the CCS build process you have changed the name of the DSP program file from the default, you must tell Hypersignal about the change before running Analog Conversion. To do this, in the System Config menu, suffix the modified filename to the ANALOG CONVERSION. For example, you might have an entries such as:

```
sigc67xx,mytest.out

sigc54xx,G726.out

dskc67xx,mpeg4.out
```

For a list of default C6xxx and C5xxx DSP executable code filenames, see section 4.1.7 above, CCS Build Output, File Types, and File Locations.

6.1.4 Probing Real-Time C Code from DirectDSP-Enabled Visual C/C++ and Visual Basic Programs

This example uses the “sim_IO” (simultaneous I/O) Visual Studio C/C++ project to show how to monitor and debug real-time C code running on the DSP. Data collected from the DSP algorithm may be graphically displayed, recorded to waveform file, or used to update GUI items such as gauges, meters, statistics and measurement text boxes, etc.

DirectDSP allows a large amount of data to be transferred from the DSP hardware to the host program with still allowing real-time I/O to continue on the DSP hardware, including interrupts from analog and digital I/O.

6.1.5 Probing Real-Time C Code from MATLAB Programs

 Copyright © Signalogic 2003-2005

6.2 Stand-Alone Mode Debugging

To debug in stand-alone mode, no change is needed in the CCS project build. The DSP executable boot-code will automatically detect the whether the system is in stand-alone mode or host mode. To cause the DSP boot code to detect and enter stand-alone mode:

3. If needed, on the CCS main menu bar, select Project | Build (or F7) to build the project.
4. Download DSP code via CCS. On the CCS main menu bar, select File | Load Program, or File | Reload Program.
5. Do not initialize or run host software, such as Hypersignal or C/C++, Visual Basic, or MATLAB program that uses DirectDSP software.
6. For SCI project builds, ensure that Program Counter register in CCS is set to zero (PC register in CCS Register View window). This should occur automatically after code download, as SCI builds define the `_c_int00` entry point as the start of the Reset vector (location zero in DSP memory). For DSP/BIOS project builds, allow Program Counter to remain pointing to DSP/BIOS-defined `_c_int00` entry point, which will be non-zero.
7.

 Copyright © Signalogic 2003-2005

6.3 Code Composer Studio Debugging Examples

Code Composer Studio offers many helpful debugging tools to assist users and developers.

Viewing Memory

Displaying a Graph of Memory

Setting Breakpoints

6.4 CCS JTAG Issues During Debug

Restarting CCS after breakpoint or instruction halt can be difficult in a real-time I/O situation. Essentially, any type of JTAG data transfer – ranging from breakpoint to memory display to register view -- will cause the processor to processor halt. This can have unintended consequences. For example, if analog I/O interrupts are occurring at best data will be lost, and at worst the analog I/O may lose sync with serial port clocks and fail to restart correctly when JTAG execution is resumed. Here are some notes about this:

1) The McBSP library InitMcBSP() API function in the SCI source code sets the FREE bit in SPCR2 register (C5xxx) or SPCR register (C6xxx) to allow serial port clocks to continue to run during a JTAG halt.

2) Low-level boot code performs a hard McBSP reset function at run-time.

3) Specific analog hardware initialization functions, such as SD4Init(), force a hard Reset to analog converters and/or audio codec chips.

During debug, the most reliable procedure to allow JTAG debug and real-time I/O to coexist is as follows:

- Try to limit number of needed breakpoints -- use judicious choice of breakpoints to halt at points where a range of relevant data values can be examined
- Remove each breakpoint after it is hit and you are ready to move past it
- Implement breakpoints in such a way that you avoid stopping unnecessarily in functions or low-level code

After changing DSP code and before restarting the DSP, the most reliable procedure to ensure a clean start is as follows:

- Perform a Reset CPU command
- Reload program (on main menu, select File, then Reload Program)
- An optional step is to reset the Program Counter (PC register in CCS register view window) to zero for C6xxx devices or 0x0080 for C5xxx devices. You may want to do this if you want to see code execution from the actual reset vector location, rather than the _c_int00 entry point – see Reset CPU Notes below for more information
- Run code again

The Reload Program step above should never be skipped. There are a few one-time variables set by low-level boot code that are required to cause proper initialization (such as detecting the

difference between Host and Stand-Alone modes). Reload Program causes these to be reset back to zero, which is needed for correct operation.

Reset CPU Command Notes

It is worthwhile to thoroughly understand the Reset CPU command in CCS software; something made all the more important because the command is not explained well in CCS documentation.

The Reset CPU command causes the DSP to perform an *internal reset* – all onchip registers are reset exactly as if the Reset line on the chip was active. However, a key distinction is that external hardware and peripherals *do not know* a DSP reset has occurred – there is no outside indication, no line that goes active, etc. SigC5xxx and SigC6xxx hardware fully takes this unusual situation into account.

CCS software takes the Reset CPU command a step further and allows the instruction(s) located at the Reset vector to execute before recapturing control over the DSP state machine. This is why the program counter typically points to the _c_int00 entry-point after the command completes. The entry point normally is at the start of low-level boot code, for example t54beg.asm or t67beg.asm boot-code for SCI builds, or an internal Texas Instruments routine such as BIOS_init() for DSP/BIOS builds.

Texas Instruments has created an application note document that describes this situation in vivid detail for C6xxx devices, using a DSK C6711 board as an example, although the information is applicable to C5xxx devices as well. It is reprinted below for additional information purposes.

Texas Instruments Application Report
SPRA978 - November 2003
TMS320C6000 Boot Mode and Emulation Reset
DSP Applications
Dipa Rao

ABSTRACT

Emulation-generated target reset, issued via a pull down menu command (Debug | Reset CPU) in Code Composer Studio (CCS) software, causes an internal chip reset. Since this reset is internal, any devices external to the DSP will not be aware of it. The /RESET signal, which is an input to the DSP, is not affected by this command. This may cause problems in a system environment where an external host or board-level system logic drives the DSP device configuration and boot pins, and if some type of host boot mode (HPI, PCI, or XBUS) is selected. Texas Instruments C6xx DSP devices have configuration pins that select the boot process after DSP reset. Depending on the boot process selected, asserting target reset through emulation can have undesired results. This document explains these conditions and the precautions that must be taken to avoid problems.

1 Hardware Reset

/RESET is an input pin on the TMS320C6xx devices. This signal forces a DSP system reset that sets internal control registers and output control signals to their default values. Refer to the TMS320C6000 CPU and Instruction Set Reference Guide (SPRU189) and TMS320C6000 Peripherals Guides for register default values. When /RESET is deasserted, its rising edge device configuration and/or boot mode pins are read and latched. These pins continue to be read for at least a few cycles after /RESET is raised high for synchronization. Please refer to the datasheet of the specific device for detailed timing information. After the boot configuration pins are latched, the appropriate boot process takes place as described in the next section. In host port boot mode, the rising edge of the /RESET signal can be used to trigger the host to start the boot process.

The target will not react to /RESET being asserted from the external system, if the emulator is plugged in and Code Composer Studio debugger has been invoked, EXCEPT for the boot logic. The following steps will illustrate this behavior.

1. Bring up Code Composer Studio on a 6711 DSK, perform an emulation generated target reset (Debug | Reset CPU), and single step a few addresses to get the PC (program counter) past 0 (zero).

2. Fill memory with a pattern beginning at address 0, press the reset button, and observe that nothing apparently happens.

3. Single step once and observe that lower memory is overwritten by the contents of Flash. The boot process was initiated but delayed because the emulation logic had control. During the single step, the boot logic was able to gain control and overwrite memory.

2 Boot Mode

After device reset, the DSP may boot code from external memory or an external device before beginning execution. The configuration pins that are read at reset determine the DSP boot process and other device configurations. The C6000 devices are capable of three types of boot modes:

- **No boot:** If this mode is selected, upon completing reset, the CPU directly begins execution from address 0 of memory. Please ensure that the CPU does not execute invalid instructions, as execution of invalid opcodes will result in undefined operations. During development stages, while using the Code Composer Studio debugger, it may be beneficial to hold down the DSP reset button (if available) until the Code Composer Studio splash screen vanishes. This minimizes the amount of time the processor runs unknown and perhaps invalid program code, thus increasing the chances of success in gaining control of the DSP for emulation.

- **Non volatile memory (ROM/Flash) boot:** When the DSP is brought out of reset in this mode, the CPU core is internally held in reset while the DMA/EDMA peripheral transfers

code from external non-volatile memory to address 0 of DSP memory. The amount of data transferred varies depending on the DSP in question. After completing the transfer, the CPU begins execution from address 0.

Typically, an unprogrammed flash device would result in the DSP boot loading 0xFFFFFFFF, which is decoded as a NOP. This by itself is not a problem; however, when the processor starts executing the instructions it could potentially continue running beyond defined memory, which could lead to problems. This might even put the processor in a state where the emulator cannot be invoked. In order to avoid this condition, it is useful, during development stage, to program the flash with at least a minimal snippet of code to ensure that the DSP emulator can be brought up reliably.

For example, a simple code snippet that can be put into the flash (at the starting address of flash) is:

```
Infinite_loop:   B .S1 Infinite_Loop
           NOP
           NOP
           NOP
           NOP
           NOP
           NOP
           NOP
```

The above sample code shows a full- packet of code that would reliably help bring up the debugger. These instructions shown above translate to the following opcodes:

```
0x00000012 Infinite_loop:    B .S1 Infinite_loop
0x00000000             NOP
```

- **Host boot:** When /RESET is de-asserted while in host boot mode, the CPU core continues to be held in reset. While the core is in reset, the external host can exercise the host interface (HPI, PCI or XBUS), access the DSP internal or external memory, and initialize the DSP internal configuration registers. After all the necessary initializations, the host sets DSPINT (host to DSP interrupt signal) to complete the boot process. This causes the CPU to wake up and start executing code at address 0. This interrupt from the host does not cause the processor to execute an interrupt service routine (ISR), since the core was in reset when it was asserted. However, the CPU must clear the DSPINT bit so that future interrupts from the host can be recognized. If the system is designed such that the host processor actually drives the boot mode pins, please ensure that every time /RESET is asserted, the host correctly configures the pins so that they can be appropriately latched.

  On older C6x0x devices, if host boot is selected, the DSP will wait for the host to initialize the DSP and set DSPINT. If the host relies on the /RESET signal to start transfer, then the host could stall indefinitely, because it never sees the /RESET signal going active during emulation reset.

  The emulation driver for C621x, C671x and C64xx devices features an internal timeout

function (the hardware to support this function is not available on C620x and C670x devices). In host boot mode, the emulator waits for a certain amount of time. When a timeout happens, the emulation logic asserts DSPINT to wake up the CPU from reset. Hence, on the C621x, C671x and C64xx devices, the emulator does not stall while waiting for the DSPINT from the external device.

3 Emulation Generated Target Reset

Emulation-generated target reset, as described here, is the reset scanned into the target from the emulator, and is initiated via a pull down menu in Code Composer Studio (Debug | Reset CPU). This reset may also be performed by issuing a command using the GEL script command GEL_Reset(). It should be noted that emulation reset is not performed on starting the Code Composer Studio debugger (unless it is directed by a GEL file). Also, please note that the emulation generated target reset being referred to in this section, is not the same as physically resetting the emulator that is being used in the development system. Unlike a hardware reset, an emulation reset causes a reset that is confined to the internals of the chip. It is not propagated outside the DSP. The system level hardware external to the DSP is not normally aware that an emulation-generated target reset is taking place. When the CPU is reset from Code Composer Studio, the boot mode pins are read, and the processor will attempt to perform the appropriate boot process, depending on the settings on the boot configuration pins. If the DSP is configured to boot from an external host, then the host will not be aware of the reset. Therefore, if the host does not respond correctly to the DSP emulation reset condition; it will not perform the boot process, which could result in emulator lock-up.

Special precautions also need to be taken if the device configuration and boot mode pins rely on an external device (such as FPGA or PLD) to drive them. This is to ensure that valid values will be read at the configuration pins during reset and the DSP does not get into an invalid boot mode.

4 Detecting Emulation Reset

Since the /RESET signal cannot be used by the host to detect emulation reset, other methods can be implemented. Some output pins do, or can be made to change states during and after reset.

For example, a timer output pin on the C621x or C671x will be in high-Z state during reset, and driven low (default) after reset. If an external pull up is hooked up to this pin, then this pin will go high during reset and low after reset. This can be used to detect an emulation-generated target reset condition. If this pin is unused and an external pull up is connected, then a rising transition of this pin can be taken as /RESET going active, and falling transition as /RESET going inactive. In general, any unused output pins in Z-group (pins that go high-Z during reset) can be externally pulled up/down opposite to the default after-reset state to detect an emulation reset. Note that the transition of this signal will be delayed with respect to the /RESET signal. Refer to device specific datasheet for pins and reset timing information. Upon detection of this condition, the host can drive the correct values to the configuration pins if necessary, and can perform the boot process that ends with asserting DSPINT.

 Copyright © Signalogic 2003-2005

5 Reset Emulator

The Reset Emulator function (Debug | Reset Emulator) should not be confused with resetting the target. Reset Emulator initializes the emulation hardware (XDS510 or XDS560) and the JTAG state machine within the target by toggling the /TRST signal. Reset Emulator should be used with caution and only in an attempt to recover from errors. If a Reset Emulator command is used, it should be accompanied by an emulation generated target reset to ensure that the target, emulation driver and emulator are in sync.

References

1. TMS320C6000 CPU and Instruction Set Reference Guide (SPRU189)

2. TMS320C620x/C670x DSP Boot Modes and Configuration Reference Guide (SPRU642)
TMS320C6000 Boot Mode and Emulation Reset

 Copyright © Signalogic 2003-2005

Hardware Issues During Debug

DMA Controller State-Machine Lockup

When the board is in a PC, it's very easy to tell if you've locked the DMA state machine, you can read from 0x32f (e.g. using debug) and if the result if 0xff then it's el-freezo.

With CCS + JTAG, you can observe the StartDMA() routine in rtc_periph.c and monitor the handshake “while” statements located in the routine. If the timeout value increments past the error limit, then the DMA state machine has not responded and is likely frozen.

 Copyright © Signalogic 2003-2005

7 DSP Software API

Peripheral access, real-time data acquisition, host interface, and signal processing function APIs are summarized here. Information about these functions which is currently distributed in several documents is being consolidated here.

7.1 Peripheral Access Functions

Sections below describe functions that may be used to access DSP and baseboard peripherals.

 Copyright © Signalogic 2003-2005

7.1.1 Digital I/O Functions

GetDigitalIOControl

unsigned char GetDigitalIOControl (uBank)

unsigned int uBank; /* digital I/O bank to control */

The GetDigitalIOControl() function returns the current direction setting for each bit of the specified I/O bank. See section 10.8 below, Digital I/O, for more information.

Parameter	**Description**
uBank	Digital I/O bank from which to read direction settings

Returns

Current direction value for each bit of the specified I/O bank. If the bank is disabled, a valid result is still returned; see SetDigitalIOControl function for more information.

Notes

1) Digital I/O is an 8-bit peripheral.
2) All bits in digital I/O control registers are always defined.

See Also

ReadDigitalIOData

SetDigitalIOControl

WriteDigitalIOData

 Copyright © Signalogic 2003-2005

ReadDigitalIOData

unsigned char ReadDigitalIOData (uBank)

unsigned int uBank; /* I/O bank from which to read data */

The ReadDigitalIOData() function reads a digital I/O bank, and can be used to retrieve individual digital input bits. See section 10.8 below, Digital I/O, for more information.

Parameter	**Description**
uBank	Digital I/O bank to read

Returns

Returns value of digital I/O bank (8-bit value). If bits are set to input, returns external value of each bit. If bits are set to output, then returns last value written by software. If bank is disabled, then results are undefined when bits are set to input, but still valid if bits are set to output. See SetDigitalIOControl function for more information.

Notes

1) Digital I/O is an 8-bit peripheral, organized in multiple banks of 8 bits.
2) Results are undefined if digital I/O banks is disabled (see SetDigitalIOControl function); otherwise, all bits in the digital I/O bank are defined.

See Also

SetDigitalIOControl

WriteDigitalIOData

 Copyright © Signalogic 2003-2005

SetDigitalIOControl

void SetDigitalIOControl (uBank, uDir, fActive)

unsigned int uBank;	/* digital I/O bank to control */
unsigned int uDir;	/* direction of each I/O bit in the bank */
BOOL fActive	/* disables or enables digital I/O bank */

The SetDigitalIOControl() function sets the direction of each bit for the specified I/O bank, and determines whether the bank is active or not. See section 10.8 below, Digital I/O, for more information.

Parameter	Description
uBank	Digital I/O bank to control
uDir	Direction of each I/O bit, corresponding 1-to-1 with bits in the digital I/O bank
fActive	Enables or disables the digital I/O bank in hardware. When digital I/O banks are disabled, the physical digital I/O lines become dedicated module site signals. Digital I/O banks correspond to module sites 1-to-1; i.e. digital I/O bank 2 is multiplexed with module site 2, etc.

Value	Meaning
DIGIO_ENABLE	Enables digital I/O bank
DIGIO_DISABLE	Disables digital I/O bank

Returns

No return value.

Notes

1) Digital I/O is an 8-bit peripheral.
2) All bits in digital I/O control registers are always defined.

See Also

GetDigitalIOControl
ReadDigitalIOData
WriteDigitalIOData

 Copyright © Signalogic 2003-2005

WriteDigitalIOData

void WriteDigitalIOData (uBank uBits)

unsigned int uBank; /* I/O bank to write data */

unsigned char uBits; /* value to write to I/O bank */

The WriteDigitalIOData() function writes to a digital I/O bank, and can be used to set individual digital output bits to one or zero. See section 10.8 below, Digital I/O, for more information.

Parameter	Description
uBank	Digital I/O bank to write
uBits	Value to write I/O bank (8-bit value).). If bits are set to output, sets external value for each bit. If bits are set to input, then there is no effect. If bank is disabled, then results are undefined. See SetDigitalIOControl function for more information.

Returns

No return value.

Notes

1) Digital I/O is an 8-bit peripheral, organized in multiple banks of 8 bits.
2) Results are undefined if digital I/O banks is disabled, and in fact may interfere with module site operation (see SetDigitalIOControl function); otherwise, all bits in each digital I/O bank are defined.

See Also

GetDigitalIOControl

ReadDigitalIOData

SetDigitalIOControl

 Copyright © Signalogic 2003-2005

7.1.2 DMA Controller Functions

There may be instances when it is necessary to access the DMA controller directly in order to move data to/from a peripheral. One example might be to write a block of data over the host bus to an external Comm card with UARTS that have FIFOs visible as a register on the bus. In such a case, it might be desirable to move a block of data from DSP memory to the external FIFO.

```
void WritePeriphData(int nDeviceType, DWORD dwOffset, void* pWord, int nLen);
int ReadPeriphData(int nDeviceType, DWORD dwOffset, void* pWord, int nLen);
```

 Copyright © Signalogic 2003-2005

7.1.3 DUART Functions

```
void WriteUARTReg(int nDeviceNum, int nReg, int nByte);
int ReadUARTReg(int nDeviceNum, int nReg);

BYTE ReadUARTStatus(int nDeviceNum);

char GetUARTChar(int nDeviceNum);
void PutUARTChar(int nDeviceNum, char cChar);

int WriteUARTData(int nDeviceNum, char* pChar, int nNumChars, BOOL fCheckStatus);
int ReadUARTData(int nDeviceNum, char* pChar, int nNumChars, BOOL fCheckStatus);

char ReadUARTChar(int nDeviceNum);           /* similar to Get/Put char, but does not wait for
char */
int WriteUARTChar(int nDeviceNum, char ch);  /* FIFO to be not empty (Read), or not full (Write)
*/

void ResetDUART();
BOOL InitUART(int nDeviceNum);

#define CHECK_STATUS   TRUE  /* parameters used in Read/Write UART data calls */
#define IGNORE_STATUS  FALSE
#define CHECK_ONCE     2
```

 Copyright © Signalogic 2003-2005

7.1.4 Flash EEPROM Functions

```
BOOL WriteFlashEEPROMData(DWORD dwAddr, unsigned int uData);
unsigned int ReadFlashEEPROMData(DWORD dwAddr);
```

 Copyright © Signalogic 2003-2005

7.1.5 Host Bus Functions

```
void WriteHostBus(unsigned int uAddr, unsigned int uVal);
unsigned int ReadHostBus(unsigned int uAddr);
```

 Copyright © Signalogic 2003-2005

7.1.6 LED Functions

LEDs may be placed either in “system mode” or in “user mode”, in which case they may be controlled by host and DSP software.

```
void SetLEDControl(unsigned char nSel);
void WriteLEDData(unsigned char nBits);
```

7.1.7 Routing Logic Functions

ReadRoutingLogic

unsigned int ReadRoutingLogic (uRegOfs)

unsigned int uRegOfs; /* offset of Routing Logic register */

The ReadRoutingLogic() function reads a register in the Routing Logic peripheral and can retrieve individual flag and bitfield values in that register. See section 10.9 below, Routing Logic, for more information.

Parameter	Description
uRegOfs	Offset of register to read in Routing Logic

Value	Meaning
RTC_TDM_RTSW_CLKCTRL	Clock source control register
RTC_TDM_RTSW_SWTB_PROC	Module serial port switch table
RTC_TDM_RTSW_SWTB_PERIPH	Peripheral switch table
RTC_TDM_RTSW_MODEMSIG	Modem signal register
RTC_TDM_RTSW_PORTCTRL	RS-422 port control register
RTC_TDM_RTSW_REVID	Revision ID register (read only)
RTC_TDM_RTSW_COMMAND	Command register

Returns

Returns the register value read as an unsigned integer (16-bit value).

Notes

1) Routing Logic is a 16-bit peripheral.
2) Undefined bits may or may not be read as zero.
3) Offset values do not need to be even values or multiplied by 2; low-level logic converts word offsets to byte offsets automatically for word peripherals.

 Copyright © Signalogic 2003-2005

See Also

SetWriteRoutingLogic

 Copyright © Signalogic 2003-2005

WriteRoutingLogic

void WriteRoutingLogic (uRegOfs, uVal)

unsigned int uRegOfs; /* offset of Routing Logic register */

unsigned int uVal; /* value to write to register */

The WriteRoutingLogic() function writes a register in the Routing Logic peripheral, and can be used to set individual flag and bitfield values in that register. See section 10.9 below, Routing Logic, for more information.

Parameter	Description	
uRegOfs	Offset of register to write in Routing Logic	
	Value	**Meaning**
	RTC_TDM_RTSW_CLKCTRL	Clock source control register
	RTC_TDM_RTSW_SWTB_PROC	Module serial port switch table
	RTC_TDM_RTSW_SWTB_PERIPH	Peripheral switch table
	RTC_TDM_RTSW_FLOWCTRL	Flow control signal register
	RTC_TDM_RTSW_PORTCTRL	RS-422 port control register
	RTC_TDM_RTSW_REVID	Revision ID register (read only)
	RTC_TDM_RTSW_COMMAND	Command register
uVal	Value to write register (16-bit value)	

Returns

No return value.

Notes

3) Routing Logic is a 16-bit peripheral.
4) Undefined bits should always be written as zero.
5) Offset values do not need to be even values or multiplied by 2; low-level logic converts word offsets to byte offsets automatically for word peripherals.

 Copyright © Signalogic 2003-2005

See Also

ReadRoutingLogic

 Copyright © Signalogic 2003-2005

8 TBD

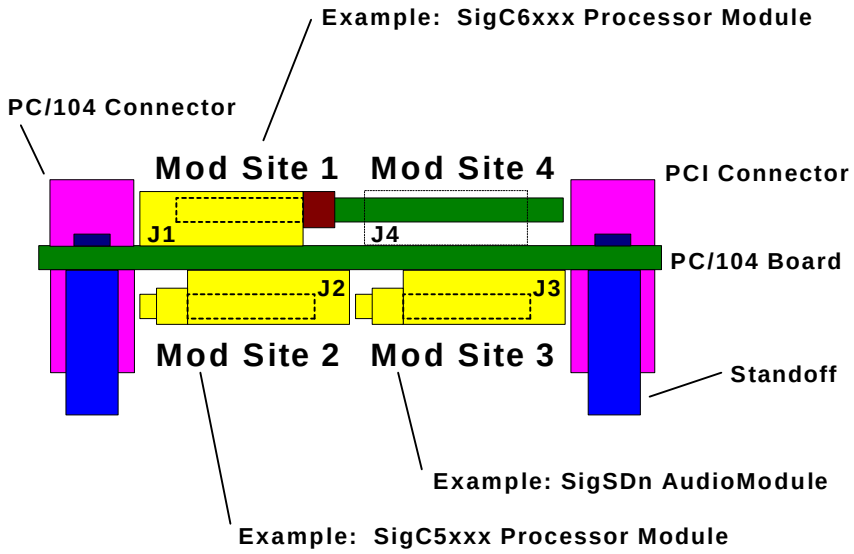
 Copyright © Signalogic 2003-2005

9 Module Interfaces

SigC5xxx and SigC6xxx board types contain module sites according to the following table.

Board Type	SODIMM 72 Sites	MiniPCI Sites
SigC5xxx PC/104, Rev C1	1 to 4	None
SigC5xxx-PC/104, Rev D	1 to 2	1 to 2
SigC6xxx PC/104, Rev C1	1 to 3	None
SigC6xxx PC/104, Rev D	1-2	1-2
SigC5xxx PCI, Rev A	1 to 4	1 to 2
SigC6xxx PCI, Rev A	1 to 4	1 to 2

Module sites are numbered as shown in the drawing below:



Notes

1) In the example shown above, module site 4 is not populated to allow the SigC6xxx processor module in module site 1 to fit without obstruction.

 Copyright © Signalogic 2003-2005

2) The J numbers shown above (J1, J2, etc) correspond to the silkscreen marking on the board. Each Jn label is printed on the board near pin 1 of the module site connector.

3) The first processor module is normally installed in module site 1. Additional processor modules are typically installed in site 4 (C5xxx) or site 3 (C6xxx).

4) When a SigC6xxx module is installed, module site 4 is usually not populated (C6xxx modules are longer due to larger processor package size).

5) When a SigSDn module is installed, module site 4 is the typical location for SigC5xxx board configurations, and module site 3 is typical for SigC6xxx board configurations.

6) Module site numbering – and which sites are populated -- is important when considering digital I/O capabilities; see section 10.8 below, Digital I/O, for more information.

7) For PC/104 boards, only Rev D and later boards provide PCI interface.

 Copyright © Signalogic 2003-2005

9.1 MiniPCI Module Interfaces

Rev D PC/104 boards and Rev A PCI boards contain MiniPCI module sites. Below are pin assignments for MiniPCI modules.

9.2 SODIMM 72 Module Interfaces

Below are pin assignments and signal descriptions for SODIMM 72 modules.

Pin No	Signal Name	I/O/Z/P	Class 1 Signal [1]	Class 2 Signal [1]	Description and Comments
1	Vcc_IO	P	Vcc_IO		3.3V typical, supplies module I/O voltage
2	GND	P	GND		
3	/HCS [3]	I	/HCS		Host bus select strobe; equivalent to module chip select
4	HA3_/HDS [2]	I	HA3_/HDS		Host bus address 3 or data strobe (module dependent). For some modules with on-module logic (e.g. CPLD), used as HA3 address signal for internal register access, and logic generates HDS internally
5	/HINT [3]	O	HINT		Module or device 0 host interrupt
6	HRDY [3]	O	HRDY		Host bus Ready signal indicating module is ready to accept or provide host bus data. For multiprocessor modules, HRDY signals from all DSPs are combined together in on-module logic to form this signal
7	/RST [3]	I	/RST		Module or device 0 Reset signal
8	TDMFS-A0	I/O/Z	TDMFS-A0		TDM/serial stream A framesync, device 0
9	GND	P	GND		
10	MCLK	I	MCLK		Module clock; distributed to individual DSPs by on-module logic
11	GND	P	GND		
12	Vcc_Core	P	Vcc_Core		2.5V typical, although may be not used, or other values depending on module and processor type. Supplies Vcc Core voltage to processors or to on-module regulators that generate Vcc Core. Example values include 1.8V, 1.6V, 1.26V, etc.
13	HA4	I	HA4		Host address 4
14	TDMTX-A0	O/Z	TDMTX-A0		Serial/TDM stream A data transmit, device 0
15	HD0	I/O/Z	HD0		Host bus data 0
16	HD1	I/O/Z	HD1		Host bus data 1
17	HD2	I/O/Z	HD2		Host bus data 2
18	HD3	I/O/Z	HD3		Host bus data 3
19	HD4	I/O/Z	HD4		Host bus data 4
20	HD5	I/O/Z	HD5		Host bus data 5
21	HD6	I/O/Z	HD6		Host bus data 6
22	HD7	I/O/Z	HD7		Host bus data 7
23	Vdd_A+	P	Vdd_A+		+5V typical, supplies positive rail for signal conditioning circuitry for audio or analog I/O modules
24	Vcc_IO	P	Vcc_IO		
25	GND	P	GND		
26	Vdd_A-	P	Vdd_A-		-5V typical, supplies negative rail for signal conditioning circuitry for audio or analog I/O modules
27	/DMAEN	O/Z	/DMAEN		DMA enable/request signal to base board
28	HR/W	I	HR/W		Host bus read/write (active low write)
29	HA0_HBIL [2]	I	HA0_HBIL		Host bus address 0 or HPI8 HBIL signal. Also used as SELA/B signal on legacy C542x modules
30	HA1_HCNTL0 [2]	I	HA1_HCNTL0		Host bus address 1 or HPI HCNTL0 signal

31	HA2_HCNTL1 [2]	I	HA2_HCNTL1		Host bus address 2 or HPI HCNTL1 signal
32	TDMTX-A1	O/Z	TDMTX-A1		Serial/TDM stream A data transmit, device 1
33	GPIO0	I/O/Z	GPIO0		General purpose I/O signal 0
34	TDMRX-B	I	TDMRX-B		Serial/TDM stream B data receive
35	GPIO1	I/O/Z	GPIO1		General purpose I/O signal 1
36	Vcc_Core	P	Vcc_Core		
37	HD8	I/O/Z	/HCS1 [4]	HD8	Host bus data 8 or device 1 chip select
38	TDMFS-B	I/O/Z	TDMFS-B		Serial/TDM stream B framesync
39	HD9	I/O/Z	/HINT1 [4]	HD9	Host bus data 9 or device 1 host interrupt
40	HD10	I/O/Z	HRDY1 [4]	HD10	Host bus data 10 or device 1 Ready signal
41	HD11	I/O/Z	/RST1 [4]	HD11	Host bus data 11 or device 1 Reset
42	TDMFS-A1	I/O/Z	TDMFS-A1		Serial/TDM stream A framesync, device 1
43	/INT0	I	/INT0		Module interrupt 0. Routed to EINT5 on C6xxx modules.
44	/INT1_BIO	I	/INT1 + BIO		Module interrupt 1 or legacy C54xx BIO signal. Routed to EINT4 on C6xxx devices.
45	TDMCLK-A	I	TDMCLK-A		Serial/TDM stream A clock
46	XF_/ACRST	O/Z	XF or Audio Codec Reset		General-purpose output flag (XF) or audio codec Reset signal (if audio module is present on the module bus)
47	TDMRX-A	I	TDMRX-A		Serial/TDM stream A data receive
48	Vcc_IO	P	Vcc_IO		
49	GND	P	GND		
50	TDMTX-A2	O/Z	TDMTX-A2		Serial/TDM stream A transmit data, device 2
51	/INT3	I	/INT3		Module interrupt 3
52	TDMTX-B	O/Z	TDMTX-B		Serial/TDM stream B transmit data
53	TDMRX-C	I	TDMRX-C		Serial/TDM stream C receive data
54	HD12	I/O/Z	/HCS2[4]	HD12	Host bus data 12 or device 2 chip select
55	TDMCLK-B	I	TDMCLK-B		Serial/TDM stream B clock
56	HD13	I/O/Z	/HINT2 [4]	HD13	Host bus data 13 or device 2 host interrupt
57	HD14	I/O/Z	HRDY2 [4]	HD14	Host bus data 14 or device 2 Ready signal
58	HD15	I/O/Z	/RST2 [4]	HD15	Host bus data 15 or device 2 Reset signal
59	TDMFS-A2	I/O/Z	TDMFS-A2		Serial/TDM stream A framesync, device 2
60	Vcc_Core	P	Vcc_Core		
61	GND	P	GND		
62	EMU1	I/O	EMU1		JTAG signal, used by Texas Instruments-compatible emulators
63	EMU0	I/O	EMU0		JTAG signal, used by Texas Instruments-compatible emulators
64	/TRST	I	/TRST		JTAG reset (according to 1149.1 standard)
65	TMS	I	TMS		JTAG Test Mode Select signal
66	TCK	I	TCK		JTAG clock
67	TDI	I	TDI		JTAG Transmit Data Input (serial format)
68	TDO	O	TDO		JTAG Transmit Data Output (serial format)
69	TDMTX-C	O/Z	TDMTX-C		Serial/TDM stream C transmit data
70	TDMCLK-C	I	TDMCLK-C		Serial/TDM stream C clock
71	MDET_TDMFS-C	I/O/Z	MDET or TDMFS-C		Serial/TDM stream C framesync. Also used as carrier board module presence detect signal at module power-up (max 50 msec)
72	GND	P	GND		

 Copyright © Signalogic 2003-2005

Notes

[1] Class 1 modules provide an 8-bit host bus interface and contain either:

on-module logic and multiple DSP devices, making no use of these signals, or

no on-module logic and multiple DSP devices, using these signals to control individual DSP devices

Class 2 modules use these signals to provide a 16-bit host bus interface and contain either:

on-module logic and multiple DSP devices, or

no on-module logic and a single DSP device

[2] Address signal for modules with onboard logic, otherwise used as strobe or control signal connected directly to individual DSP devices.

[3] Signal applies if module contains logic (e.g. CPLD), otherwise applies to device 0.

[4] Data signal for modules with HPI16 capability, otherwise used as individual device select or other signals for modules with no onboard logic.

+ Indicates signals are wired-OR'ed together.

Table entry is left blank if signal does not directly relate to a DSP signal.

I/O/Z/P refers to input, output, high-impedance (tri-state), or pwr/gnd.

Indicates signals required for 8-bit host processor interface.

Indicates additional signals required for 16-bit host processor interface.

Indicates signals typically used for interface to external TDM format serial device; examples include RS-422, T1/E1, etc.

Indicates analog signals for modules containing integrated audio codecs. Please refer to connector page excerpt from the module schematic.

9.3 Module Host Processor Interface

The following signals (shown shaded as light orange in sections 9.1 and 9.2 above) are required to implement an 8-bit "host processor interface", referred to here as HPI8. This is commonly called a CPU or microcontroller interface in many applications.

 Copyright © Signalogic 2003-2005

HD[7:0]	Host bi-directional data or input address (whether bus contains data or address depends on value of HA0-HA2)
HA0_HBIL	Either host address signal for on-module logic internal registers, or host low byte / high byte indicator for HPI8 interfaces to 16-bit C54xx and C55xx devices
HA1_HCNTL0	Either host address signal for on-module logic internal registers, or host control signal to select register in DSP onchip HPI register bank
HA2_HCNTL1	Either host address signal for on-module internal logic registers, or host control signal to select register in DSP onchip HPI register bank
HR/W	Host read/write control
/HCS	Host signal to select module (equivalent to module chip select)
/HDS	Host data strobe
HRDY	Host Ready signal (indicates module ready condition)

The following additional signals (shown as shaded in light green in sections 9.1 and 9.2 above) are used to provide a 16-bit host interface, referred to here as HPI16:

HD[15:8]	Host bi-directional data or input address (whether bus contains data or address depends on value of HA0-HA2)

The following optional signal may be used to interrupt a host processor to indicate that DSP software requires servicing. This signal is not required to implement an 8-bit or 16-bit HPI; an alternative approach is for host software to poll locations in DSP memory, or a combination of the two approaches.

/HINT	Host interrupt (active low, edge-triggered)

 Copyright © Signalogic 2003-2005

HPI Access Sequence

To access module data, the host processor sequence is typically as follows:

1) Host asserts /HCS, HA0-HA2, and HR/W
2) Host waits for HRDY to indicate "ready" condition
3) Host asserts /HDS
4) Host reads or writes data on HD0-HD7
5) Host de-asserts /HDS
6) Host optionally de-asserts /HCS if other devices on the host bus need to be accessed
7) Host repeasts steps 1-6, as needed

Note that "HPI" in the above signal descriptions refers to "Host Port Interface", which is a peripheral commonly found on Texas Instruments DSP devices that allows external processors to directly access internal memory of the DSP. HPI is a "slave" peripheral; i.e. the DSP cannot activate the strobe signals under DSP software control.

HPI timing and some signal polarities (e.g. HRDY) may vary depending on the type of DSP device on the module. For example, timing of C54xx devices is not the same as C67xx devices. Detailed timing diagrams are given in the following documentation and online references for specific DSP types:

C54xx HPI

Valid for SigC54xx modules.

TMS320C54x DSP Reference Set
Volume 5: Enhanced Peripherals
Chapter 4

http://focus.ti.com/lit/ug/spru302/spru302.pdf

C5510 HPI

Valid for SigC5510 and SigC5502 modules.

TMS320VC5510 DSP Host Port Interface (HPI) Reference Guide

http://focus.ti.com/lit/ug/spru588b/spru588b.pdf

C6000 HPI

Valid for SigC67xx, SigC62xx, and SigDM642 modules.

 Copyright © Signalogic 2003-2005

TMS320C6000 DSP Host Port Interface (HPI) Reference Guide

http://focus.ti.com/lit/ug/spru578c/spru578c.pdf

 Copyright © Signalogic 2003-2005

10 Board Resources and Peripherals

Basic documentation for register level access is given in sections below. ***Note that Signalogic does not support customers at this level, instead providing off-the-shelf device drivers and example programs – both host software and DSP software – for this purpose.***

The documentation in this section is provided as-is. It tends to be updated slowly and frequently contains inaccuracies. If you are writing your own driver or otherwise trying to access DSP devices, peripheral devices, or DMA controller at register level using your own code, ***then you are on your own.*** That means Signalogic engineering staff will not spend time to help you figure out low-level chip details – for example Texas Instruments errata for host port auto-increment bugs -- when our engineers have already spent weeks or months on such problems. If this is not acceptable, then please ask management level personnel at your company to contact Signalogic for additional discussion.

 Copyright © Signalogic 2003-2005

10.1 Host Interface

The host interface allows an external or host processor to “see” and communicate with onboard DSPs, logic, and peripherals.

There are two types of host interface provided by SigC5xxx and SigC6xxx boards:

- PC104 interface in I/O address space
- PCI interface

PC104 and PCI host interfaces are explained in more detail in the following sections.

 Copyright © Signalogic 2003-2005

10.1.1 PC104 Host Interface

Host software "sees" the standard PC104 board interface as a single, 16-byte region in the host PC I/O address space. An overall view of DSP processor cores, logic registers, and peripheral registers accessible via the PC104 host interface is shown below:

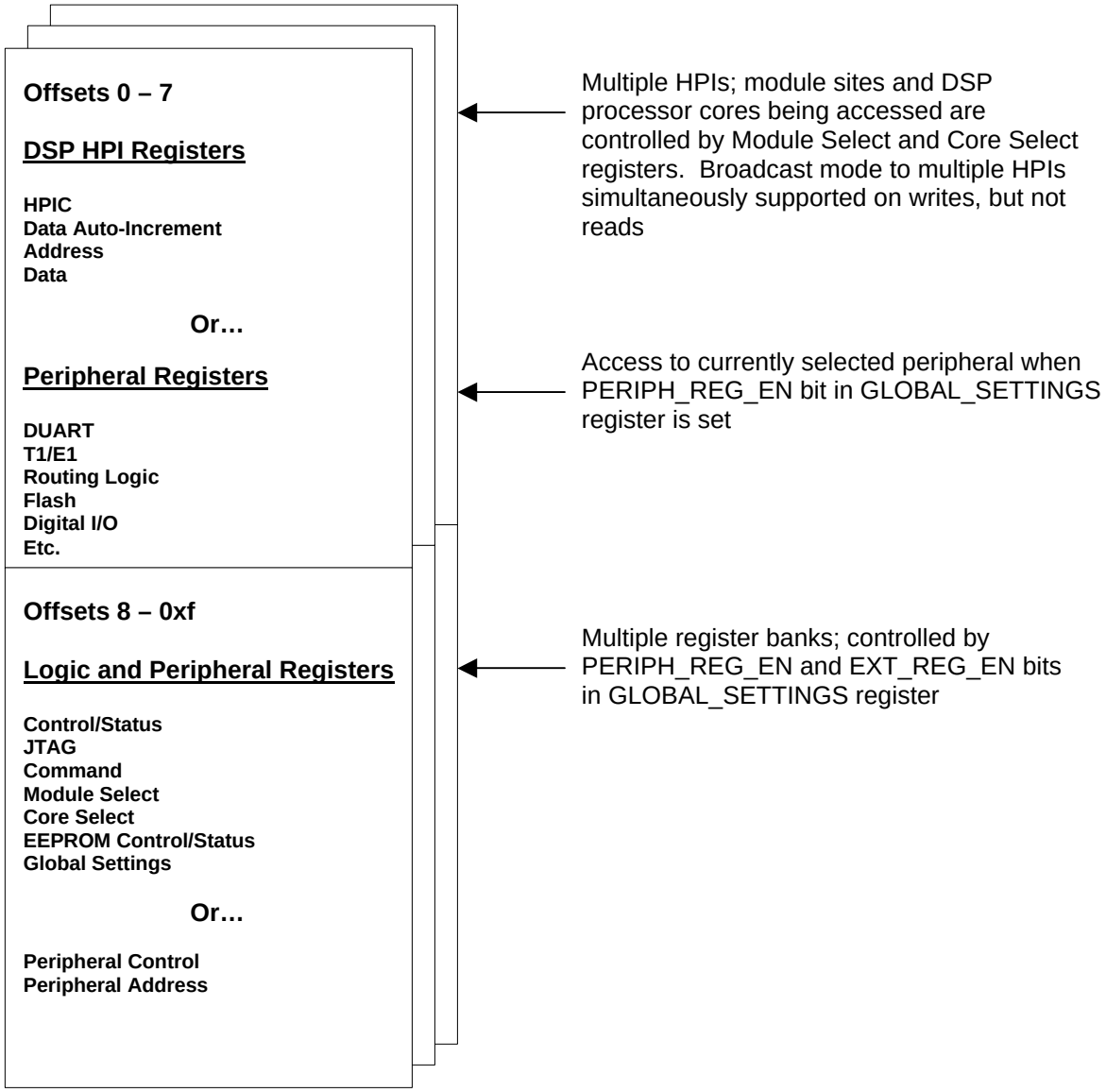


Figure 10-1, PC104 Host Interface Overview

 Copyright © Signalogic 2003-2005

Host PC104 Interface Notes

1. Register offsets use 8-bit addressing.
2. Registers are typically 8 and 16 bits wide. To see 16-bit register values, word I/O access must be used (see Global Settings Register for more information).

 Copyright © Signalogic 2003-2005

10.1.2 PCI Host Interface

Host software “sees” the PCI host interface as three (3) separate memory-mapped areas:

- “high performance” BAR that allows high data rate burst mode and bus master transfers to one or more DSPs
- “buffer” BAR area used for burst mode and bus master transfers
- “compatibility mode” BAR (Base Address Register) that includes all register sets available in the PC104 Host Interface (see section 10.1.1 above)

An overall view of DSP processor cores, logic registers, and peripheral registers accessible via the PCI host interface is shown below in Figure 10-2.

 Copyright © Signalogic 2003-2005

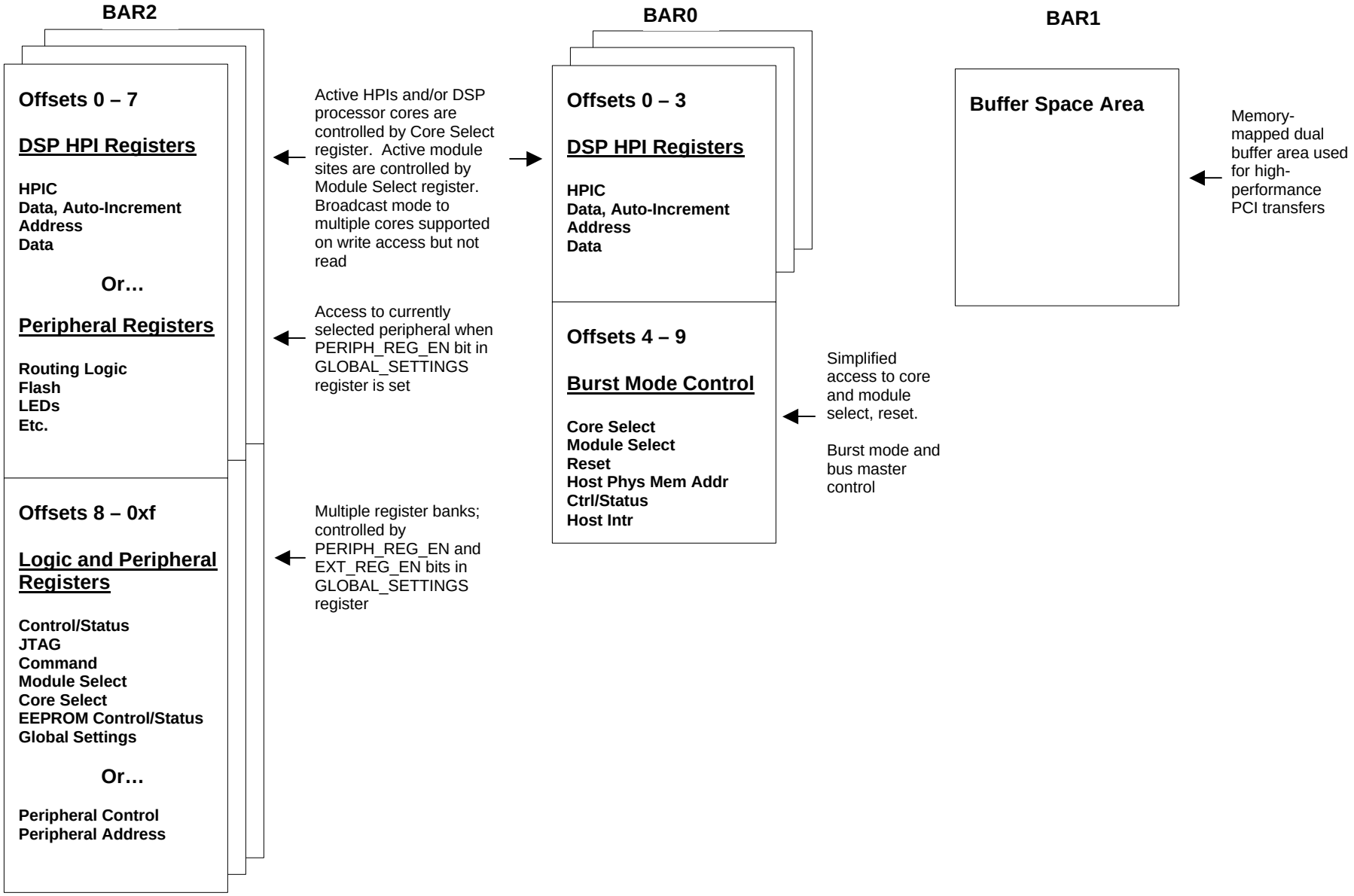


Figure 10-2, PCI Host Interface Overview

Host PCI Interface Notes

1. All register offsets use 32-bit addressing.
2. BAR2 registers are typically 8 and 16 bits wide; BAR0 and BAR 1 registers are 32-bits wide.

 Copyright © Signalogic 2003-2005

10.1.2.1 PCI Configuration Space Values

Below are current PCI configuration space values for SigC5xxx and SigC6xxx PCI boards. Note that this information is subject to change.

Field	Value	Comments
Vendor ID [1]	0x10EE	Xilinx PCI core logic value
Device ID [1]	0x5190	Xilinx PCI core logic value
Class Code	0x0B40	Generic Co-Processor
Rev ID	0x0090	Logic revision v0.90
Subsystem Vendor ID	0x10EE	Vendor ID repeated
Subsystem Device ID	0x5190	Subsystem Device ID, repeated

Notes

[1] Temporary value, subject to change.

10.1.2.2 BAR0 Description

Following are PCI register definitions for BAR0:

Register Name	32-Bit Offset	DSP Accessible	Comments
HPIC	0	Y	HPI Control/Status register
HPIA	1	Y	HPI Address
HPIDI	2	N	HPI Data, with auto-increment
HPID	3	N	HPI Data, no auto-increment
CTRL_STATUS	4	N/A	Control/Status register
MODSEL	5	N/A	Module Select register
CSEL	6	N/A	Core/Chip Select register
CSEL_EX1	7	N/A	Core/Chip Select register extension 1
CSEL_EX2	8	N/A	Core/Chip Select register extension 2

 Copyright © Signalogic 2003-2005

RESET	9	N/A	Reset register
DMA_HOST_ADDR	10	N/A	Host memory physical address register
DMA_HOST_CTRL	11	N/A	DMA control register, host side
DMA_DSP_CTRL	12	N/A	DMA control register, DSP side
DMA_FLAGS	13	N/A	DMA flags and status register
INTR_STATUS	14	N/A	Interrupt register

HPIC, HPIA, HPIDI, and HPID Registers

The HPIC, HPIA, HPIDI, and HPID registers map to HPI registers inside each DSP. When one or more CSEL bits for DSPs are active, the host driver has direct access to HPI of the active DSPs.

Only single-PCI transfer access is permitted to HPI registers. Access to these registers stalls the PCI bus while the EDMA of the selected DSP services the transaction. If more than one DSP is active, then the PCI bus must wait for the slowest EDMA.

For more information on HPI registers, see section 2.3, TMS320C64x HPI16 or HPI32, in the Texas Instruments document TMS320C6000 DSP Host Port Interface (HPI) Reference Guide (TI document number SPRU578B).

CTRL_STATUS Register

RESERVED (31 -- 8)	X	HHWIL	CORE_RST	PROC_RST	INT3	INT1	INT0	X

The Ctrl/Status register contains different DSP control and status bits. Only single-PCI transfer access is permitted to this register. Ctrl/Status register bits are defined as follows:

INT0

DSP INT0 enable. Currently not implemented.

INT1

DSP INT1 enable. Currently not implemented.

INT3

DSP INT3 enable. Currently not implemented.

PROC_RST

Processor Reset bit. When the PROC_RST bit is set (or cleared), the edge change causes ones (or zeros) to latch into the RESET register for DSPs that have their corresponding bits set in the CSEL register. If the DSP's CSEL bit is zero, there would be no change in its Reset register bit.

See below for more information on the RESET and CSEL registers.

CORE_RST

Core Reset bit. Currently not implemented.

HHWIL. Currently not implemented.

MODSEL Register

RESERVED (31 -- 8)	M7 – M0

The Module Select register allows groups of DSPs to be enabled/disabled. Some boards support actual physical "module sockets", other boards support banks of DSPs. When an Mn bit is set, all operations on the DSPs in that module is permitted, including core select, memory data transfer, reset, etc. When an Mn bit is clear, the DSPs in the module are held in Reset and no DSP operations are permitted.

Only single-PCI transfer access is permitted to this register.

Notes

1. For SigC6415-PTMC cards, only M0 is active. M1-M7 are reserved and should not be accessed, but if written then written as zero.

2. For SigC5561-PTMC cards, only M0 and M1 are active (each bank contains up to 6 DSPs). M2-M7 are reserved and should not be accessed, but if written then written as zero.

CSEL, CSEL_EX1, and CSEL_EX2 Registers

C31	C30 -- C3	C2	C1	C0

C63	C62 -- C35	C34	C33	C32

C95	C94 -- C67	C66	C65	C64

CSEL registers use bit fields to select which DSP HPI ports are currently active. Memory access and reset register operations use the current CSEL setting to determine on which cores on which devices to operate.

A '1' bit indicates a core is active during memory access, reset, and other processor operations, and a '0' indicates a core is not active.

Only single-PCI transfer access is permitted to this register.

Notes

1. SigC64xx boards contain single-core DSPs, so each Cn bit corresponds to a chip-select signal. For SigC6415-PTMC cards, only C0-C7 are active. C8-C95 are reserved and should not be accessed, but if written then written as zero.

2. SigC55xx boards contain multi-core DSPs, so every N Cn bits map to a chip-select signal. For example, for SigC5561-PTMC card, C0-C5 map to a chip select signal for DSP0. For SigC5561-PTMC cards, only C0-C11 are active. C12-C95 are reserved and should not be accessed, but if written then written as zero.

3. Multi-core DSP operation is not implemented in logic versions 0.9x.

4. Internally, the logic does not allow multiple chip-select signals to be active during HPI reads. If software driver calls are made such that more than one chip-select signal is active at the time of HPI read, the logic activates only the chip-select corresponding to the first set lsb in the CSEL registers. For example, a CSEL register setting of 0x000A would reduce to 0x0002.

RESET Register

RST31	RST30 -- RST3	RST2	RST1	RST0

The Reset Register allows direct control of Reset lines for individual DSPs. Clearing a bit will place the corresponding DSP into Reset, and setting a bit will release the DSP from Reset.

Only single-PCI transfer access is permitted to this register.

Notes

3. For SigC6415-PTMC cards, only RST0-RST7 are active. RST8-RST31 are reserved and should not be accessed, but if written then written as zero.

4. For SigC5561-PTMC cards, only RST0-RST11 are active. RST12-RST31 are reserved and should not be accessed, but if written then written as zero.

DMA_HOST_ADDR Register

HA31 – HA0

After host application layer software allocates a memory buffer to receive bus-mastering transfer data from the SigC5/6xxx board, it must pass the memory buffer "logical address" to the host driver. The driver must then determine the actual host system memory "physical address" and make it known to SigC5/6xxx board PCI logic, by setting the DMA_HOST_ADDR. This must be done prior subsequent bus mastering transfer, initiated by writing to the DMA_CTRL register.

Only single-PCI transfer access is permitted to this register.

DMA_HOST_CTRL Register

DIR	BUFSEL	RESERVED (29 – 10)	LEN (9 – 0)

The DMA_HOST_CTRL register allows the host driver to control and initiate bus-mastering transfers. Writes to the register initiate a bus-mastering transfer with the control values written.

Only single-PCI transfer access is permitted to this register.

Following are descriptions of individual DMA_HOST_CTRL register bits.

DIR

DIR is the direction indicator. If DIR is cleared, the SigC5/6xxx board will transfer data from the host system memory to its local transfer buffer. If DIR is set, the SigC5/6xxx board will transfer data from its local buffer into host system memory.

BUFSEL

BUFSEL is the buffer selection indicator. If BUFSEL is cleared, the transfer will be to/from Buffer0 inside the SigC5/6xxx board's local transfer buffer memory area. If BUFSEL is set, the transfer will use Buffer1.

LEN

LEN specifies the bus master transfer length as the number of 32-bit words. The current maximum transfer length is 1024.

 Copyright © Signalogic 2003-2005

DMA_DSP_CTRL Register

DIR	BUFSEL	RESERVED (29 – 10)	LEN (9 – 0)

The DMA_DSP_CTRL register allows DSP code to control and initiate bus-mastering transfers. Writes to the register by a DSP initiatee a bus-mastering transfer with the control values written.

Following are descriptions of individual DMA_DSP_CTRL register bits.

DMA_FLAGS Register

PE	SSE	RMA	RTA	STA	DST	DPE	FBBC	0	66MHZ	CL	IS	RSRVD (18 – 8)	MA	TA	DWD	DWOD	TT	NT	TE	DT

SigC5/6xxx board PCI interface logic can detect differing methods of transaction termination including no-error. The DMA_FLAGS register provides access to those detection codes, serving as driver integrity and diagnostic tool.

Only single-PCI transfer access is permitted to this register.

Following are DMA_FLAGS register bit definitions; all values are active high (1 == active) and given from the perspective of SigC5/6xxx board PCI logic.

PE

Parity Error Detected

SSE

Signaled System Error

RMA

Received Master Abort

RTA

Received Target Abort

STA

Signaled Target Abort

DST

 Copyright © Signalogic 2003-2005

Device Select Timing

[10] –Slow
[01] – Medium

DPED

Data Parity Error Detected

FBBC

Fast Back-to-Back Capable, Always 0

Bit22

Always 0

66MHZ

66 MHz Capable, Always 0

CL

Capabilities List

IS

Interrupt Status

MA

Master Abort

TA

Target Abort

DWD

Disconnect With Data

DWOD

Disconnect Without Data

 Copyright © Signalogic 2003-2005

TT

Target Termination

NT

Normal Termination

TE

Transaction End

DT

Data Transfer

INTR_FLAGS Register

RESERVED (31 – 1)	BMTC

The Interrupt Flags register allows SigC5/6xxx logic to signal interrupts to the host driver by setting bits on a per-interrupt basis. The host driver may clear an interrupt by writing a '1' to the corresponding bit.

After reacting to a SigC5/6xxx board interrupt, the host driver must clear the corresponding interrupt bit to prevent further, indefinite interrupts from occurring. The host driver may read the INTR_FLAGS register value and write it back to clear all devices asserting interrupts.

Only single-PCI transfer access is permitted to this register.

Following are bits defined in the INTR_FLAGS register:

BMTC

Interrupt indicating that bus mastering transfer has completed.

10.1.2.3 BAR1 Description

BAR1 maps buffer space used for high-performance mode PCI transfers to/from one or more DSP HPI(s).

Byte offset 0x0000 – 0x0FFF is defined as Buffer0 and 0x1000 – 0x1FFF is as Buffer1.

 Copyright © Signalogic 2003-2005

10.1.2.4 BAR2 Description

BAR2 memory map follow the register offset and functional definition given in sections 10.2 through 10.4 below. BAR2 provides a “compatibility mode” interface to allow existing software utilities, diagnostic and instrumentation and measurement software to function with only a driver change. Examples include register-level board test suite and Hypersignal-Macro software.

 Copyright © Signalogic 2003-2005

10.2 DSP HPI Register Access

DSP HPI access from host and from DSP software is enabled via logic interface on the hardware. From the base I/O address (ISA bus) or BAR0 (PCI bus), the first 8 locations correspond to HPI registers in the C5xxx or C6xxx device. Note that registers are byte-addressable to support C5xxx device HPI8 access, but also may be accessed as words on word boundary to support HPI16 device access.

For more information on PCI bus access, see section 10.1.2 above, PCI Interface.

Note that for DSP devices which are target-only (e.g. HPI port only, with no bus master capability), a DMA controller implemented in logic is used to allow DSP devices to access board peripherals, external host bus, and other DSP devices via their HPI ports.

Example host and DSP C code identifiers for DSP HPI accesses are given below.

C Code Identifier	**Offset**	**Comments**
DSP_CTRL_REG_HI	0	DSP HPIC register high byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, access as word for C5xxx and C6xxx devices in HPI16 mode.
DSP_CTRL_REG_LO	1	DSP HPIC register low byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, ***do not access*** for C5xxx and C6xxx devices in HPI16 mode.
DSP_DATA_REG_HI_AINC	2	DSP data auto-increment register high byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, access as word for C5xxx and C6xxx devices in HPI16 mode.
DSP_DATA_REG_LO_AINC	3	DSP data auto-increment register low byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, ***do not access*** for C5xxx and C6xxx devices in HPI16 mode.
DSP_ADDR_REG_HI	4	DSP address register high byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, access as word for C5xxx and C6xxx devices in HPI16 mode.

 Copyright © Signalogic 2003-2005

DSP_ADDR_REG_LO	5	DSP address register low byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, ***do not access*** for C5xxx and C6xxx devices in HPI16 mode.
DSP_DATA_REG_HI	6	DSP data register high byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, access as word for C5xxx and C6xxx devices in HPI16 mode.
DSP_DATA_REG_LO	7	DSP data register low byte. See TI documentation for detailed information. Access as byte for C5xxx devices in HPI8 mode, ***do not access*** for C5xxx and C6xxx devices in HPI16 mode.

Detailed information about these registers can be found in Texas Instruments C54xx, C55xx, and C6xxx series documentation, under sections labeled HPI and Enhanced HPI. Below is a list of Texas Instruments books in which to look:

C54x CPU and Peripherals
C54x Enhanced Peripherals
C6000 Peripherals

 Copyright © Signalogic 2003-2005

10.3 Logic Registers

Logic register access from host and from DSP software is enabled via logic interface on the hardware. From the base I/O address (ISA bus) or BAR0 (PCI bus), the second 8 locations (8 through 15) are mapped to logic registers when the PERIPH_REG_EN bit in the GLOBAL_SETTINGS register is set to 0.

Note that registers are byte-addressable, and in some cases accessible as words on word boundary, in order to support both byte-addressable and word-addressable peripherals.

Note that for DSP devices which are target-only (e.g. HPI port only), a DMA controller is used to allow DSP access to board peripherals, external host bus, and other DSP device HPI ports.

Below is additional information about logic registers.

 Copyright © Signalogic 2003-2005

Logic Registers

Host and Module Interface

Offset †	Main Registers	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8	CTRL_STATUS [1]	DMA_ACTIVE	SYS_RESET_N	CORE_RESET_N ‡	PROC_RESET_N	INT0	INT1_BIO	INT3	JTAG_SEL
9	JTAG_CTRL_STATUS	LPBK_EN	TMS	TCK	TDI	TDO	RSRVD	RSRVD	OUTPUT_EN ‡
0xa	COMMAND [2]	M_INT_REG_EN	HI_BYTE_EN	RESERVED					
0xb	MODULE_SEL [3]	MOD8_EN	MOD7_EN	MOD6_EN	MOD5_EN	MOD4_EN	MOD3_EN	MOD2_EN	MOD1_EN
0xc	CORE_SEL_LO [2, 4]	CORE7_EN	CORE6_EN	CORE5_EN	CORE4_EN	CORE3_EN	CORE2_N	CORE1_EN	CORE0_EN
0xd	CORE_SEL_HI [2, 4]	EE_BOOT_STAT	RESERVED			SER_DATA_OUT	SER_DATA_IN	DEVICE_CS	SER_CLK
0xe	EE_I2C_DEVICE [6]	PERIPH_REG_EN	DSP_8/16 [7]	HOST_8/16 [7]	DMA_EN	RESERVED	INT1_BIO_EN [7]	INT3_EN [7]	EXT_REG_EN [8]
0xf	GLOBAL_SETTINGS [5]								

General Notes

† Offset is relative to base I/O address (ISA bus) or BAR (PCI bus). All offsets assume 16 available locations.

‡ "_N" is used to denote an active low signal. "EN" is used as an abbreviation for "Enable".

All bits are read/write unless otherwise denoted by (RO) or (WO) designators. Registers with "STATUS" suffix contain one or more hardware inputs that may be read and monitored.

All register bits not defined are reserved and should be written as a zero. Undefined bits may not read as zero.

Register Specific Notes

[1] For modules with on-module logic, CTRL_STATUS and CORE_SEL registers are reflected in on-module logic. In this case, writes cause both baseboard and module logic registers to be updated, reads access only module logic registers.

[2] COMMAND register bits 5-0 are reserved and should be left unused if legacy hardware must be supported.

[3] MODULE_SEL register enables one or more module sites or DSP banks, depending on the board type.

[4] CORE_SEL register bits select processors for boards/modules with single-core devices. CORE_SEL register is 16-bits deep when HOST_8/16 is zero. The upper 8 bits may be accessed by odd byte reads when HOST_8/16 is 1. CORE_SEL register is 32-bit deep when EXT_REG_EN bit is active; used for example on SigC544x boards.

[5] GLOBAL_SETTINGS register visible when bit 7 (PERIPH_REG_EN) is zero; Peripheral Register bank is visible when bit 7 is one.

[6] EE_I2C_DEVICE allows host and DSP interface to I2C bus on board, or other serial bus devices, such as serial EEPROM. In some cases other EEPROM devices use this register; EE_BOOT_STAT provides indication of valid Flash EEPROM boot.

[7] See Reset Conditions notes.

[8] EXT_REG_EN on some boards enables additional registers.

Reset Conditions

DSP_8/16 and HOST_8/16 are set hy module type auto-detection logic at power-up, as follows:

Module Type	DSP_8/16	HOST_8/16	INT3_EN	INT1_BIO_EN
No Module	1	1	1	0
C549 (legacy)	1	1	0	0
C54xx	1	1	0	0
C55xx	0	1	0	0
C6xxx	0	1	0	0

SigC54/55/6xx Logic Register Definitions

Rev H2, May 2004
Copyright (C) Signalogic, 2002-2004

Revision History

Created: Apr 2002
Revised: Sep 2002, added peripheral control
Revised: Nov 2002, T1/E1 and DUART additions
Revised: Jan 2003, C6xxx additions
Revised: Jun/Jul 2003, Flash and I/O enables, misc updates
Revised: Aug 2003, additional notes and corrections
Revised: Nov 2003, digital I/O and Flash EEPROM corrections
Revised: Dec/Jan 2004, External Bus Master Support
Revised: Mar 2004, TDM Routing Logic Improvements
Revised: Apr 2004, Added Power Control register, made LED register banks a peripheral
Revised: May 2004, added EE_BOOT_STAT, INT1_BIO_EN, SYS_RESET

 Copyright © Signalogic 2003-2004

Logic Register Notes

1. For modules with on-module logic, CTRL_STATUS and CORE_SEL registers are reflected in on-module logic. In this case, writes cause both baseboard and module logic registers to be updated, reads access only module logic registers. Examples of this case include the SigC67xx and SigC5561 module series. An example of a module design with no on-module logic is the SigC54xx module series.

2. COMMAND register bits 5-0 are reserved and should be left unused if legacy hardware must be supported.

3. MODULE_SEL register enables one or more module sites or DSP banks, depending on the board type.

4. CORE_SEL register bits select processors for boards/modules with single-core devices. CORE_SEL register is 16-bits deep when HOST_8/16 is zero. The upper 8 bits may be accessed by odd byte reads when HOST_8/16 is 1. CORE_SEL register is 32-bit deep when EXT_REG_EN bit is active; used for example on SigC544x boards.

5. GLOBAL_SETTINGS register visible when bit 7 (PERIPH_REG_EN) is zero; Peripheral Register bank is visible when bit 7 is one.

6. The EE_I2C_DEVICE register allows host and DSP interface to a PC104 or PTMC board I2C bus, or other serial bus devices, such as serial EEPROM. In some cases other EEPROM devices use this register; as an example, the EE_BOOT_STAT bit provides indication of valid Flash EEPROM boot.

7. See Reset Conditions notes in Logic Registers graphic above.

8. EXT_REG_EN on some boards enables additional registers. See board-specific documentation.

Specific Register Notes

Below are additional notes about specific registers.

Ctrl/Status Register

1) Default value of the Control/Status register at power-up or system reset is dependent on module type auto-detection logic that executes at logic boot-time, and also on whether the stand-alone mode jumper (JP14 on PC/104 boards) is installed. Depending on the module type installed, possible power-on / system reset values of the Control/Status register are shown below:

 Copyright © Signalogic 2003-2005

Module Type	Value if Stand-Alone Mode Jumper Installed	Value if Stand-Alone Mode Jumper Not Installed
No Module	0	0
C549 (legacy)	0x1e	0
C5409A	0x1e	0
C5416	0x1e	0
C5421	0x3e	0
C55xx	0x3e	0
C6xxx	0x3e	0

2) DSP processor Reset signals are maintained by internal registers that are not directly accessible to host or DSP software, but are indirectly accessible. Internal reset registers are latched ***only on transitions*** of the Reset bit in the Control/Status register, and depending on current value of the Core Select register. For example, to reset one or more processors, the following sequence should be followed:

- set Core Select Register (see below) to zero; no processors are selected
- write 1 to PROC_RESET_N bit in Control/Status Register (and/or CORE_SELECT_N bit if DSP is multi-core device)
- select desired processors in Core Select Register
- write 0 to Reset bit in Control/Status Register

After the above sequence, the selected processors will be held in Reset.

3) Processors may be released from the reset state as needed, either all at once or in any combination, as needed. The same procedure as above should be followed, except sequence of writes to the PROC_RESET_N bit should be 0-1.

4) It is recommended to hold unused processors in reset, to both minimize power consumption on the module and to avoid possible situations where a processor runs undefined code and adversely affects other processors.

5) For multicore devices such as C5421 and C5441 the PROC_RESET_N bit applies to HPI Reset signals and the CORE_RESET_N bit is used for individual cores. For single-core C5xxx devices, only the PROC_RESET_N bit is active.

Module Select Register

1) Default value of the Module Select Register at power-up is zero. Module type auto-detection logic that runs at logic boot-time may set this register to 1.

 Copyright © Signalogic 2003-2005

2) Each bit in the Module Select Register enables a module site on the board, as shown above. More than one module may be selected at one time.

3) For DSP Farm type boards without module sites, each bank of 8 DSPs is considered a "module". There are a maximum of 32 cores in a module or bank.

Core Select Register

1) Default value of the Core Select Register at power-up is zero. Module type auto-detection logic that runs at logic boot-time may set this register to 1.

2) For modules with multicore processors each bit represents a core. For example, the SigC5421 module contains eight (8) C5421 processors with two (2) cores each; in this case, the Core Select Register is 16 bits wide and each bit represents a core.

3) For modules with on-module CPLD logic, this register is "reflected" inside the CPLD logic and actually exists there, and not in the PC104 or PTMC board logic.

4) For modules with less than N processors or cores, bits 31 .. N-1 in the Core Select Register are not used.

5) Each bit in the Core Select Register enables a processor in currently selected module site(s). Multiple processors may be selected at once; in this case the standard Signalogic host software drivers can perform "broadcast" writes. ***Read operations should only be attempted with one processor selected at a time to avoid contention between DSP HPI ports.***

Global Settings Register

1) Default value of the Global Settings register at power-up or system reset is dependent on module type auto-detection logic that executes at logic boot-time. Depending on the module type installed, possible power-on / system reset values of the Global Settings register are shown below:

Module Type	Value
No Module	0x62
C549 (legacy)	0x62
C54xx	0x60
C55xx	0x20
C6xxx	0x20

2) The PERIPH_REG_EN bit enables onboard peripherals to be enabled at addresses base + 0 to base + 7, instead of DSP modules. See section 10.4 below, Peripheral Registers.

 Copyright © Signalogic 2003-2005

3) DSP_8/16 determines whether DSP module accesses are 16-bit ore 8-bit. Default value is set according to module type auto-detection logic; see table above.

4) Host_8/16 determines whether host bus accesses are handled as 8-bit or 16-bit. Default value is set according to module type auto-detection logic; see table above. This bit must be set prior to making 16-bit register accesses from host software drivers.

5) DMA_EN enables the DMA controller logic in test/debug mode. This bit should not be set by customers under any circumstances.

6) CLK_DIV enables division by 2 of the master logic clock for internal usage. This bit is used for test and special modes by Signalogic and should not be set by customers under any circumstances.

Example host and DSP C code identifiers for logic register accesses are given below. When developing DSP code applications, see the file rtc_periph.h for more information.

C Code Identifier	**Offset**	**Comments**
CTRL_STATUS_REG	8	Control/Status Register
JTAG_REG	9	JTAG Register
CMD_REG	0xa	Command Register
MOD_SEL_REG	0xb	Module Select Register
CORE_SEL_REG	0xc	Core Select Register
GLOBAL_SETTINGS_REG	0xf	Global Settings Register

 Copyright © Signalogic 2003-2005

10.4 Peripheral Registers

Peripheral register access from host and from DSP software is enabled via logic interface on the hardware. From the base I/O address (ISA bus) or BAR0 (PCI bus), the second 8 locations (8 through 15) are mapped to peripheral registers when the PERIPH_REG_EN bit in the GLOBAL_SETTINGS register is set to 1.

Note that registers are byte-addressable, and in some cases accessible as words on word boundary, in order to support both byte-addressable and word-addressable peripherals.

Note that for DSP devices which are target-only (e.g. HPI port only), a DMA controller is used to allow DSP access to board peripherals, external host bus, and other DSP device HPI ports.

Below is additional information about peripheral registers.

 Copyright © Signalogic 2003-2004

Peripheral Registers

I/O and Peripheral Device Interface

Offset †	Peripheral Registers	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8	PERIPHERAL_CTRL_LO [3]	LED_EN [1]	T1E1_RST_N ‡	FLASH_RST_N	FLASH_EN ‡	T1E1_EN [2]	DUART_EN [2]	DUART_RST_N [2]	TDM_RTSW_EN [2]
9	PERIPHERAL_CTRL_HI [3]	RESERVED		EXT_BUS_EN [4]	IO_EN [5]	IO_BANK3_EN	IO_BANK2_EN	IO_BANK1_EN	IO_BANK0_EN
0xa	PERIPHERAL_ADDR_0_LO [3]	PA10-PA3 [6]							
0xb	PERIPHERAL_ADDR_0_HI [3]	PA18-PA11							
0xc	PERIPHERAL_ADDR_1_LO [3]	PA25-PA19							
0xd	PERIPHERAL_ADDR_1_HI [3]								
0xe	POWER_CTRL [7]	RESERVED	RESERVED	PA31-PA26					
0xf	GLOBAL_SETTINGS [8]	RS_422_EN	RS_232_EN	RESERVED		CLK_DIV			
		PERIPH_REG_EN	DSP_8/16 [9]	HOST_8/16 [9]	DMA_EN	RESERVED	INT1_BIO_EN [9]	INT3_EN [9]	EXT_REG_EN

General Notes

† Offset is relative to base I/O address (ISA bus) or BAR (PCI bus). All offsets assume 16 available locations.

‡ "_N" is used to denote an active low signal. "EN" is used as an abbreviation for "Enable".

All bits are read/write unless otherwise denoted by (RO) or (WO) designators. Registers with "STATUS" suffix contain one or more hardware inputs that may be read and monitored.

All register bits not defined are reserved and should be written as a zero. Undefined bits may not read as zero.

Register Specific Notes

[1] LED_EN controls LED banks on the board. When LED_EN is set, register offsets 4-7 address LED CTRL registers in which each bit selects whether the LED is under user-control (1) or system control (0). Register offsets 0-3 address LED DATA registers, in which each bit selects LED value (1=on, 0 =off) when that LED is under user-control. Four (4) banks of 8 LEDs each are addressable, although some boards may have less.

[2] Registers in other peripherals (DUART, T1E1, Ethernet PHY, etc) are documented in their respective data sheets. For TDM Routing and Switch Table peripheral, selected by TDM_RTSW_EN bit, see following pages for additional documentation.

[3] PERIPERAL_CTRL and PERIPHERAL_ADDRn registers are 16-bits deep when HOST_8/16 is zero. The upper 8 bits may be accessed by odd byte reads when HOST_8/16 is 1. CORE_SEL register is 32-bit deep when EXT_REG_EN bit is active; used for example on SigC544x boards.

[4] EXT_BUS_EN enables through-accesses on expansion header. Through-accesses are independent from digial I/O bits (see note 5).

[5] When IO_EN is set, register offsets 0-3 address banks of digital I/O bits on the expansion header, and addresses 4-7 access I/O direction-control bits for each bank. For direction-control bits, a value of 0 sets the bit to input, 1 sets the bit to output.

[6] PA2-PA0 are equivalent to registers 0-7; i.e. accessing registers 0-7 when any peripheral enable bit is set will activate PA2-PA0 at time of peripheral read/write.

[7] The Power Control register may be used to help control board power consumption. RS_422_EN and RS_232_EN bits control power to RS-422 and RS-232 drivers, respectively; these bits are set to 1 at power-on to allow drivers to operate but may be set to 0 when drivers are not needed. CLK_DIV is set to 0 at power-on (no clock division), but may be used to divide the processor clock by a value from 2 to 16. Processor(s) should execute an IDLE instruction after setting a CLK_DIV register value.

[8] GLOBAL_SETTINGS register visible when bit 7 (PERIPH_REG_EN) is zero; Peripheral Register bank is visible when bit 7 is one.

[9] See Reset Conditions notes in Logic Registers description.

 Copyright © Signalogic 2003-2004

Peripheral Register Notes

1. LED_EN controls LED banks on the board. When LED_EN is set, register offsets 4-7 address LED CTRL registers in which each bit selects whether the LED is under user-control (1) or system control (0). Register offsets 0-3 address LED DATA registers, in which each bit selects LED value (1=on, 0 =off) when that LED is under user-control. Four (4) banks of 8 LEDs each are addressable, although some boards may have less.

2. Registers in other peripherals (DUART, T1E1, Ethernet PHY, etc) are documented in their respective data sheets. For TDM Routing and Switch Table peripheral, selected by TDM_RTSW_EN bit, see following pages for additional documentation.

3. PERIPERAL_CTRL and PERIPHERAL_ADDRn registers are 16-bits deep when HOST_8/16 is zero. The upper 8 bits may be accessed by odd byte reads when HOST_8/16 is 1. CORE_SEL register is 32-bit deep when EXT_REG_EN bit is active; used for example on SigC544x boards.

4. EXT_BUS_EN enables through-accesses on expansion header. Through-accesses are independent from digital I/O bits (see note 5).

5. When IO_EN is set, register offsets 0-3 address banks of digital I/O bits on the expansion header, and addresses 4-7 access I/O direction-control bits for each bank. For direction-control bits, a value of 0 sets the bit to input, 1 sets the bit to output.

6. PA2-PA0 are equivalent to registers 0-7; i.e. accessing registers 0-7 when any peripheral enable bit is set will activate PA2-PA0 at time of peripheral read/write.

7. The Power Control register may be used to help control board power consumption. RS_422_EN and RS_232_EN bits control power to RS-422 and RS-232 drivers, respectively; these bits are set to 1 at power-on to allow drivers to operate but may be set to 0 when drivers are not needed. CLK_DIV is set to 0 at power-on (no clock division), but may be used to divide the processor clock by a value from 2 to 16. Processor(s) should execute an IDLE instruction after setting a CLK_DIV register value.

8. GLOBAL_SETTINGS register visible when bit 7 (PERIPH_REG_EN) is zero; Peripheral Register bank is visible when bit 7 is one.

9. See Reset Conditions notes in the graphic in Logic Registers description above (in section 10.3 above, Logic Registers).

System LED Usage Definition

When LED_CTRL bits are zero LEDs will reflect system values, as shown below:

 Copyright © Signalogic 2003-2005

LED	Description of System Value	Comments
LED0 (CR1 on board silkscreen)	Indicates logic boot status	1 for boot Ok, 0 or flashing if error
LED1 (CR2 on board silkscreen)	Indicates stand-alone mode initialization status	1 for stand-alone initialization Ok, 0 or flashing if not

When LED_CTRL bits are 1, LEDs will reflect user values. In this case, LED_DATAn values determine LED state, under user control. A 1 value for bit n will illuminate the LEDn. See note 1 above.

 Copyright © Signalogic 2003-2005

10.5 DMA Controller

The DMA controller provides a method of moving data between DSPs and peripherals without host processor intervention. For PTMC boards, the DMA controller is also actively involved in burst mode and bus master transfers.

The DMA controller can be accessed both by host software and by DSP software. For DSP software, it provides the main method of allowing one or more DSPs to transfer data to/from other DSPs and to access onboard peripherals.

For host software, the DMA controller is less often used; for PTMC boards DMA controller functions are utilized transparently, depending on the type of API-level call made by host application software, and what driver-level functions that are invoked as a result. From both host and DSP perspectives, control register bits are accessible in the CTRL_STATUS and GLOBAL_SETTINGS registers.

10.5.1 DMA Controller Testing

10.5.2 Initial DMA Configuration

In DSP code, an initial DMA controller configuration statement is required at the start of void main(), as shown below:

```
SetDevice(RTC_DEV_ALL, RTC_DEV_ENABLE_RELEASE | DEV_DISABLE);
```

The above line is a good idea for any situation where JTAG debug is being used. It disables all I/O peripherals and releases any "enable hold" pending internally to the peripheral I/O library. The intent is to force the peripheral library -- and to some extent DMA controller logic -- to a known state regardless of what point JTAG emulation was stopped during the previous debug session.

10.5.3 DMA Controller Performance

The logic currently handles burst transfers between DSP and DUART, routing logic, and Flash -- burst mode, or transferring multiple values per transaction, is by far the most efficient.

In the example test code, the ReadDigitalIOData() and WriteDigitalIOData() function calls mentioned in section 3 above are currently invoked in "synchronous mode", where they wait for the DMA transaction to complete. All six (6) DMA transactions (digial I/O accesses) inside the real-time frame loop in melp_rt example C code require about 360 usec total in synchronous mode. This time is spent by the peripheral library waiting to return the caller (blocking), which may eventually cause an issue in a real-time system that is tight on MIPS resources. To make transactions faster, asynchronous mode can be used, which causes the peripheral library to not wait for DMA transactions to complete. When asynchronous mode is active, *only one* call to

 Copyright © Signalogic 2003-2005

Read or Write should be made during each frame loop, so there should be a counter that increments from 0 to 5 and starts over. For example:

```
/* assumes count, bank2_io, bank3_io, and bank4_io are set to zero */

   if (count == 0)
      bank2_io = ReadDigitalIOData(RTC_IO_BANK2) & 0x18;
   else if (count == 1)
      bank3_io = ReadDigitalIOData(RTC_IO_BANK3) & 0x0f;
   else if (count == 2)
      bank4_io = ReadDigitalIOData(RTC_IO_BANK4) & 0xe0;
   else if (count == 3)
      WriteDigitalIOData(RTC_IO_BANK2, bank2_io);
   else if (count == 4)
      WriteDigitalIOData(RTC_IO_BANK3, bank3_io);
   else if (count == 5)
      WriteDigitalIOData(RTC_IO_BANK4, bank4_io);

   count++;
   if (count == 5) count = 0;
```

The advantage in this situation is that each ReadDigitalIOData() or WriteDigitalIOData() function call only requires less than 500 nsec. However, panel I/O latency becomes dependent on the real-time frame loop period and the number of digital I/O accesses. In the above structure, there would be an approximate 135 msec delay to update all panel I/O when running at 2400 bps and an approximate 400 msec delay when running at 1200 bps. A latency less than 1/2 second should be acceptable for a user-interface; if not, the above code can be made "smarter" to adjust the count variable such that the I/O bank needing update (output), as a result of user input, is accessed during the next frame loop. It should be possible to always maintain a latency under 70 msec using that approach.

Also, for future system consideration, a version of DMA controller logic is under development that will allow a single call to ReadDigitalIOData() and WriteDigitalIOData() to update from 1 to 4 digital I/O banks at one time. This will improve the performance of synchronous mode operation, as well as simplifying application level code to some extent.

 Copyright © Signalogic 2003-2005

10.6 DUART

Note: the DUART peripheral is supported by PC104 boards only.

10.6.1 DUART Testing

10.6.2 DUART Performance Notes

A key DUART performance issue to be aware of the UART FIFO size, which is 128 char maximum. For example, in the speech case, buffer-ready occurs every 22 msec and the UART baud rate is 19.2 kHz, or about 2k chars/sec, so approx 10 char can be written every buffer. As buffer size increases, you can conceivably write a number of chars up to the FIFO maximum.

Note that this works in both synchronous and async modes: in sync mode the peripheral library waits about 60 usec per UART write, or 600 usec per buffer, in async mode the WriteDUARTData() function writes all 10 chars in one transaction and does not wait.

UART I/O access performance may be increased by sacrificing status checking, and/or carefully structuring application code. Here are some examples:

- use "CHECK_ONCE" or "IGNORE_STATUS" attributes when calling ReadUARTData() or WriteUARTData()

- use async mode when calling same

CHECK_ONCE means the calls will check UART FIFO status for the first char, and if nothing is there (or for Tx if the FIFO is full) then return 0. Otherwise they will transfer the specified number of chars without checking status again. IGNORE_STATUS means no checks are made. Not checking UART status is of course, faster, because the DMA controller makes no status check transactions.

Async mode has to be used carefully. You do not want to make another DMA controller transaction while not allowing the first one to complete. See items 6) and 7) below.

Some example DUART access and loopback code is located in main.c, within the _PERIPH_TEST_ compiler directives. In this code, there are examples of both char read/write, and also "bulk transfer" read, both in sync mode and async mode.

 Copyright © Signalogic 2003-2005

10.7 Flash/Boot EEPROM

Note: both PC104 and PTMC boards contain Flash/boot EEPROM. In addition:

- some PC104 boards support DSP modules with optional on-module boot/Flash EEPROM
- PTMC boards contain optional NAND Flash EEPROM for multimedia data storage

10.7.1 Flash EEPROM Testing

10.7.2 EEPROM Programming Instructions

To program Flash EEPROM on the SigC5xxx and SigC6xxx PC/104 boards, there are three (3) main steps:

- Create ROM file (.rom) from DSP executable (.out) file
- Program Flash EEPROM on PC104 or PTMC board using host PC utility program
- Configure board for stand-alone operation and reset the board

10.7.2.1 Creating a ROM File

To create a .rom file, follow the steps below:

1) Use Signalogic-provided .out file (DSP executable file) or create .out file using CCS software.

2) Run mkrom.bat, using one of the examples shown below for different DSP processor types. Note that the .out file used as input in the examples may be named differently, depending on your application and purposes:

   Example for board with C54xx processor(s):

   ```
   mkrom –54 tmsc54x.out
   ```

   Example for board with C55xx processor(s):

   ```
   mkrom –55 tmsc55x.out
   ```

   Example for board with C6xxx processor(s):

   ```
   mkrom –6 tmsc6x11.out
   ```

The mkrom.bat batch file takes in a .out file created by Texas Inst Code Composer Studio (CCS) software and creates a .rom file. Mkrom.bat generates the following default .rom filenames:

Processor Type	Default .rom Filename
C54xx	tmsc54x.rom
C55xx	tmsc55x.rom
C6x1x	tmsc6x1x.rom
C64xx	tmsc64x.rom

Mkrom.bat invokes TI's hex500 (C5xxx processors) or hex6x (C6xxx processors) "hex conversion" utility (provided with CCS software), using the Tektronix "extended 32-bit addressin" format. 32-bit addressing allows unified support for both C6xxx and C5xxx devices, as well as C5xxx support in cases of loading code into far memory (above 64k). Note that mkrom expects a standard installation of TI tools and CCS software; for example, hex500 and hex6x should be typically located at:

```
\ti\c5400\cgtools\bin
```

or

```
\ti\c\6000\cgtools\bin
```

If not, then mkrom.bat can be edited to point to the correct path as needed using a standard text editor.

10.7.2.2 Programming Flash EEPROM

To program the Flash EEPROM on the Signalogic PC104 or PTMC board, run the digio_test.exe host PC utility program. Note that this step assumes you either have the Sig board inserted into a PC using a passive carrier card, or the board is external to the PC and you have a PC104-to-ISA adapter cable installed. In either case, the Flash EEPROM utility program assumes the Sig board is accessible via PC104 or PCI bus.

The digio_test program is a combined digital I/O test and EEPROM programming utility. At the initial start prompts, enter:

Entry	Description	Comments
2	Board Type	2 = PC/104 3 = PCI
2	Processor Type	2 = C5xxx 3 = C6xxx

1	Module Site	Module site to be used for initial DSP boot
5	Test Type	Flash EEPROM erase, test, and program

The utility will first erase and test the EEPROM, and if no error occurs, it will prompt you for the following:

Prompt		**Comments**
Shared Prompts	.rom filename	Either filename or full path can be entered; if only filename, the utility searches on the current subdir and the hsmacro subdir
C5xxx Specific Prompts	PMST Value	The PMST register sets memory map, interrupt table, and Overlay mode values for C54xx processors. For C5409 the default value is 0xe0, for C5416 the default value is 0xa8; entry is expected in hex ('0x' may or may not be entered). See additional notes below.
	SWWSR	Not currently implemented. Default value in DSP after reset is assumed.
	BCSR	Not currently implemented. Default value in DSP after reset is assumed.
C6xxx Specific Prompts	EMIF Global Control Register (GBLCTL)	The GBLCTL register controls overall EMIF settings for C6xxx processors.
	SDRAM Control Register (SDCTL)	The SDCTL register controls, among other things, number of SDRAM row and column address lines and DRAM refresh timing. Default value is 0x00003778.
	CE0 Address Space Control Register (CE0CTL)	The CE0CTL register controls CE0 address space settings. Default value of 0xffffff30 assumes SDRAM in CE0 space, starting at DSP memory address 0x80000000
Shared Prompts	Processor Core List	Default value is 1; each bit indicates a core to receive Flash data at boot time and to be released from reset. For example, a core list of 7 would indicate to boot the first 3 cores on the board

Following are some processor-specific notes about prompt entry listed above.

C5xxx Processor Notes

1. Processor specific prompts only appear depending on the value entered for the initial Processor Type prompt when the programming utility first runs. Shared prompts always appear, regardless of processor type.

2. The programming utility stores the above e parameters and others in a separate "header" area in the Flash EEPROM. The header area includes signature data that allows boot logic to detect whether a valid program has been stored in Flash EEPROM. For detailed information on header area format, as well as other EEPROM area format, see section 10.7.4 below, EEPROM Storage Format.

3. The PMST value should be 0x00e0 for C5409 or C5409A devices, and 0x00a8 for C5416 or C5421 devices. Whatever value you enter in the EEPROM programming utility will be deposited at location 0x3ffc1 in the EEPROM, i.e. the 2nd location of the boot program. The first instruction of the boot program is:

```
PMST = #0xNNNN
```

where NNNN is the value entered in the programming utility prompt.

4. Boot code in t54beg.asm first sets PMST to 0x00e0, and then set back to 0x00a8 if the processor type auto-detection code finds a C5416 or C542x processor.

C6xxx Processor Notes

1. Default values for SDRAM row and column addresses are 12 and 8.

10.7.2.3 Configuring the Board for Stand-Alone Operation

Configuring the Signalogic PC104 or PTMC board for stand-alone operation will cause logic on the board to test the Flash EEPROM for a valid program, verify that auto-detected module type matches the stored program type, and boot the DSP(s) from the Flash EEPROM program. To configure the board for stand-alone operation, jumper JP14 must be inserted.

After verifying JP14 is correctly installed, the board must be reset or run from power-on for the configuration to take effect.

Depending on the size of the boot program, there should be a delay between the 2 LEDs, up to about 1/2 sec. The stand-alone-mode LED (top one, farthest from PC/104 connectors) should light first, the "board fully initialized" LED will light after the EEPROM program is completely transferred to DSP memory.

 Copyright © Signalogic 2003-2005

10.7.3 EEPROM Boot Flow

It is important to understand the boot process. The flow diagram below helps to explain:

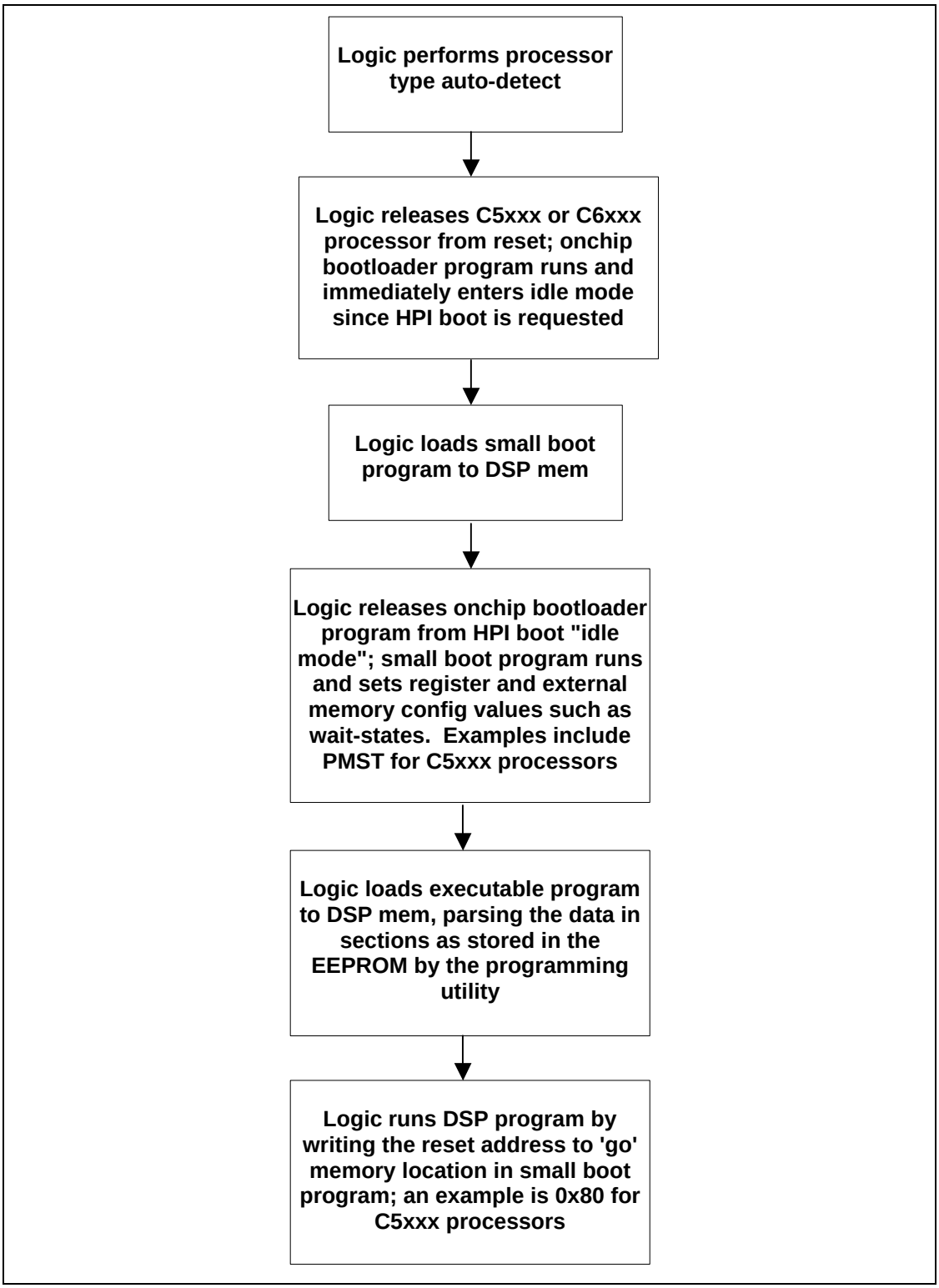

Logic performs processor type auto-detect
Logic releases C5xxx or C6xxx processor from reset; onchip bootloader program runs and immediately enters idle mode since HPI boot is requested
Logic loads small boot program to DSP mem
Logic releases onchip bootloader program from HPI boot "idle mode"; small boot program runs and sets register and external memory config values such as wait-states. Examples include PMST for C5xxx processors
Logic loads executable program to DSP mem, parsing the data in sections as stored in the EEPROM by the programming utility
Logic runs DSP program by writing the reset address to 'go' memory location in small boot program; an example is 0x80 for C5xxx processors

Figure 10-3, EEPROM Boot Flow

 Copyright © Signalogic 2003-2005

Note that the above flow allows key values, such as PMST for C5xxx processors, to be set ***prior*** to loading executable program and data. This is vitally important, as the settings for areas such as data memory above 0x8000 for C5xxx processors and external SDRAM for C6xxx processors can be critical to allowing correct program and data structure load.

The small boot program is a 32-word program, temporarily loaded at 0x1000 which sets the registers and then waits in an idle loop for the logic to load the executable and tell it to run. When done loading, the logic writes the desired reset vector to a "go flag" location in the small bootload program, causing it to jump to the reset vector. The rationale and function of the logic boot program is similar to Texas Instruments' onchip bootloader programs, except tailored for HPI mode and the SigC5xxx-PC/104 board architecture.

Note that the small boot program occupies an area normally reserved for the host interface "talker" program, so this area can safely be used in stand-alone applications.

10.7.4 EEPROM Storage Format

Below is a map of showing the format of data stored in Flash EEPROM by the programming utility:

Location	**Description**	**Contents**	**Comments**
0	Signature	0x1234	All values have to match or EEPROM boot will not occur
1	""	0x5678	
2	""	0xaaaa	
3	""	0x5555	
4	Module Type	0 = none 1 = C549 2 = C54xx 3 = C6x11	Set by user-entry for Processor Type prompt in EEPROM programming utility
5	Module List	1 to 4	Only valid if Module Type != 0
6	Core List	1 to 0x3f	Each bit specifies a DSP core; for example C5416 is a single-core processor so each bit in effect specifies a processor, but C5441 is a quad-core processor, so the first core on the second processor would be specified by bit 4

 Copyright © Signalogic 2003-2005

7 - 0x7e	Reserved for future Signalogic use	0	
0x7f	Data Format Version	0x0100	v1.0
0x80 – 0x3ffbf	Executable DSP code or non-volatile data storage	Application program	If code, then notes apply: -should be contents of .rom file, created from .out file by mkrom.bat (uses hex500 or hex6x utilities) -code starting address (reset vector) expected to be 0x80 If data, then format is application-specific; Read/Write Flash EEPROM APIs in rtc_periph.c should be used
0x3ffc0 - 0x3ffff	Executable DSP code	Boot program contains C5xxx processor values for PMST, SWWSR, XWSR, BCSR, etc, or C6xxx processor values for SDRAM configuration such as number of row and column address lines. See above EEPROM Programming Instructions for detailed information	

Notes

1. The boot program is 32 instructions long. The last location is a “poll” location used to release the boot program from idle mode. As noted above in Figure 10-3 above, EEPROM Boot Flow, boot logic writes a non-zero value to the poll location to cause the boot program to branch to the CPU Reset vector.

2. The boot program is loaded in DSP memory at 0x1000. Note that this implies the executable DSP program loaded from EEPROM cannot write over 0x1000 area.

 Copyright © Signalogic 2003-2005

3. Note that each C6xxx instruction is addressed on 4-byte boundaries (count by 4), and each C5xxx instruction is addressed on single word boundary (count by 1). Therefore, the address of the last location in the boot program in DSP memory is 0x107c for C6xxx processors, and 0x101f for C5xxx processors.

4. If you use the Flash EEPROM for non-volatile data storage, for example to store data from a DSP program, it is suggested that data storage start no lower than EEPROM location 0x20000. This will leave ample space for program storage (approx 256k x 16 for C5xxx devices, and approximately 128k x 32 for C6xxx devices). In any case, you should be very careful not to overwrite valid EEPROM executable program data. Note that a DSP program can learn the total length of stored program data by reading the section lengths and in the EEPROM until the last length, which will be zero. See the following section.

 Copyright © Signalogic 2003-2005

10.7.5 Executable DSP Program Storage Format in EEPROM

The format of executable DSP program storage in Flash EEPROM follows.

Section 1 Address Low
Section 1 Address High
Section 1 Length (M1)
Data 0
Data 1
.
.
Data M1-1
Section 2 Address Low
Section 2 Address High
Section 2 Length (M2)
Data 0
Data 1
.
.
Data M2-1
.
.
Section N-1 Address Low
Section N-1 Address High
Section N-1 Length (M3)
Data 0
Data 1
.
.
Data M3-1
Section Address Low (zero)
Section Address High (zero)
Section Length (zero) ← *last section indicated by NULL length*

Notes

1. In the above depiction, N is number of sections and Mn is the data length for each section.

2. Each EEPROM data word is 16-bits (the Flash EEPROM is a 16-bit peripheral, each address increment by 1 indicates a 16-bit word).

3. For C6xxx devices, 2x data words per instruction are read from EEPROM and written to DSP memory.

4. For C5xxx devices, Section Address High is non-zero only when extended program memory pages are used (i.e. a far-mode executable file has been used for mkrom input). In this case, boot logic automatically sets XHPIA as needed to correctly store data using 24-bit addressing.

5. Section length for last section is zero, causing boot load to stop.

10.7.6 EEPROM Boot Program Listings

Below are program listings for the small boot programs stored in Flash EEPROM and initially loaded into DSP memory by PC104 or PTMC board boot logic. The boot programs are necessary to allow DSP code to initialize critical registers such as PMST (C5xxx) and GBLCTL and SDTCL (C6xxx) that may not be reachable via HPI, prior to trying to access external memory or other DSP resources.

10.7.6.1 C5xxx EEPROM Boot Program Listing

```
TMS320C54x COFF Assembler         Version 3.70     Tue Jun 01 21:16:12 2004
Copyright (c) 1996-2001           Texas Instruments Incorporated
bootload.asm                                                     PAGE    1

       1              ********************************************************************************
       2              ; bootload.asm:  code loaded by onboard logic in stand-alone mode
       3              ;
       4              ; Copyright (C) 2003, Signalogic
       5              ;
       6              ; Created:  Sep03, JHB
       7              ;
       8              ; boot program:  loaded at 0x1000 by onboard logic before processor is released from reset; 0x1000
       9              ;                is where C54xx bootloader jumps to in HPI mode.  Performs following functions:
      10              ;
      11              ;                1) set PMST register to a default value (main purpose is to allow user control
      12              ;                   over this value (if needed), via Flash EEPROM programming utility)
      13              ;
      14              ;                2) idles, waits for host bootloader to set "go" location at 0x1010
      15              ;
      16              ;                3) jump to code reset location at 0x80
      17              ;
      18              ;                Note that it is important to set PMST *prior* to logic continuing to load code and
      19              ;                data.  Otherwise, information, for example data structures, may not be correctly
      20              ;                placed.
      21
      22                        .mmregs
      23
      24        002B  XSWCR   .set       2Bh          ; additional software wait-state control reg
      25
      26 000000               .text
      27
      28              ;         PMST = #0x0e0
      29 000000 771D          PMST = #0x0a8
         000001 00A8
      30 000002 F495          nop
      31 000003 F495          nop
      32 000004 F495          nop
      33 000005 7728          SWWSR = #0x0249       ; Insert wait state for offchip memory
         000006 0249
      34 000007 F495          nop
      35 000008 F495          nop
      36 000009 772B          mmr(XSWCR) = #0       ; set XSWCR (no ws multiplier)
         00000a 0000
      37 00000b F495          nop
      38 00000c F495          nop
      39 00000d 7729          BSCR = #2             ; initial setting OK for C549, C5409A, C5416
         00000e 0002
      40 00000f F495          nop
      41 000010 F495          nop
      42
      43 000011       wait_reset:
      44 000011 10F8          A = *(0x101f)
         000012 101F
      45 000013 F495          nop
      46 000014 F495          nop
      47 000015 F495          nop
      48 000016 F845          if (AEQ) goto wait_reset
         000017 0011'
      49
      50 000018 F073          goto #0x80
         000019 0080
      51
      52 00001a F495          nop
      53 00001b F495          nop
      54 00001c F495          nop
      55 00001d F495          nop
      56 00001e F495          nop
      57
      58              ; note:  "go" location should be 0x101f -- don't move from this address
      59
      60 00001f       go:
      61
      62 00001f 0000          .word 0
      63
      64                      .end
```

```
No Assembly Errors, No Assembly Warnings
```

10.7.6.2 C6xxx EEPROM Boot Program Listing

```
TMS320C6x COFF Assembler PC Version 4.32 Sun May 16 18:14:49 2004

Tools Copyright (c) 1996-2002 Texas Instruments Incorporated
bootload.asm                                                     PAGE    1

       1                    ******************************************************************************
       2                    ; bootload.asm:  code loaded by onboard logic in stand-alone mode
       3                    ;
       4                    ; Copyright (C) 2004, Signalogic
       5                    ;
       6                    ; Created:  Apr04, JHB, RP
       7                    ;
       8                    ; boot program:  loaded at 0x1000 by onboard logic before processor is released from reset; 0x1000
       9                    ;                is where C54xx bootloader jumps to in HPI mode.  Performs following functions:
      10                    ;
      11                    ;                1) set SDRAM registers to a default value (main purpose is to allow user control
      12                    ;                   over this value (if needed), via Flash EEPROM programming utility)
      13                    ;
      14                    ;                2) idles, waits for host bootloader to set "go" location at 0x107c
      15                    ;
      16                    ;                3) jump to code reset vectorlocation at 0x0000
      17                    ;
      18                    ;                Note that it is important to set critical SDRAM setup registers *prior* to logic
      19                    ;                continuing to load code and data.  Otherwise, information, for example data
      20                    ;                structures, may not be correctly placed.
      21
      22                             .global  _c_int00
      23
      24 00000000                    .text
      25
      26 00000000           _c_int00:
      27
      28                     ; initialize GBLCTL (Global Control) register
      29
      30 00000000 0100002A           MVKL   0x01800000, B2
      31 00000004 0100C06A           MVKH   0x01800000, B2
      32
      33 00000008 01187C28           MVKL   0x000030F8, A2  ; note that actual value loaded into A1 is set by Flash programming
ut
      34 0000000c 01000068           MVKH   0x000030F8, A2
      35
      36 00000010 010802F4           STW    A2, *B2
      37
      38                     ; initialize SDCTL (SDRAM Control) register
      39
      40 00000014 00800C2A           MVKL   0x01800018, B1
      41 00000018 0080C06A           MVKH   0x01800018, B1
      42
      43 0000001c 009BBC28           MVKL   0x00003778, A1  ; note that actual value loaded into A1 is set by Flash programming
ut
      44 00000020 00800068           MVKH   0x00003778, A1
      45
      46 00000024 008402F4           STW    A1, *B1
      47
      48                     ; initialize CE0CTL (CE0 Control) register
      49
      50 00000028 0100042A           MVKL   0x01800008, B2
      51 0000002c 0100C06A           MVKH   0x01800008, B2
      52
      53 00000030 017F9828           MVKL   0xFFFFFF30, A2  ; note that actual value loaded into A2 is set by Flash programming
ut
      54 00000034 017FFFE8           MVKH   0xFFFFFF30, A2
      55
      56 00000038 010802F4           STW    A2, *B2
      57
      58
      59                     ; loop and wait for board controller to release
      60
      61 0000003c 00883E2A           MVKL   0x107c, B1
      62 00000040 0080006A           MVKH   0x107c, B1
      63
      64 00000044           wait_reset:
      65 00000044 008402E4           LDW    *B1, A1                 ; read mem location
      66 00000048 00006000           NOP 4
      67
      68 0000004c 90000092           [!A1]  B wait_reset            ; continue until all onchip read at least once (note:  in host
      69 00000050 00008000           NOP 5
      70
      71 00000054 008428C0           SUB    A1, A1, A1
      72 00000058 00041362           B      A1                      ; jump to reset vector at location 0x000, run user program
      73 0000005c 00008000           NOP 5
      74
      75 00000060 00000000           NOP 1
      76
      77
      78                     ; note:  "go" location should be 0x107c -- don't move from this address
      79
```

 Copyright © Signalogic 2003-2005

```
      80 00000064           go:
      81
      82 00000064 00000000          .word 0
      83
      84                            .end

No Assembly Errors, No Assembly Warnings
```

10.8 Digital I/O

Note: the Digital I/O peripheral is supported by PC104 boards only.

The Digital I/O peripheral provided by SigC5xxx and SigC6xxx hardware is organized as four (4) banks of 8 bits each, or up to 32 bits total. Digital I/O bits are physically available on the Expansion Header (J5), as shown at left in the board picture below.

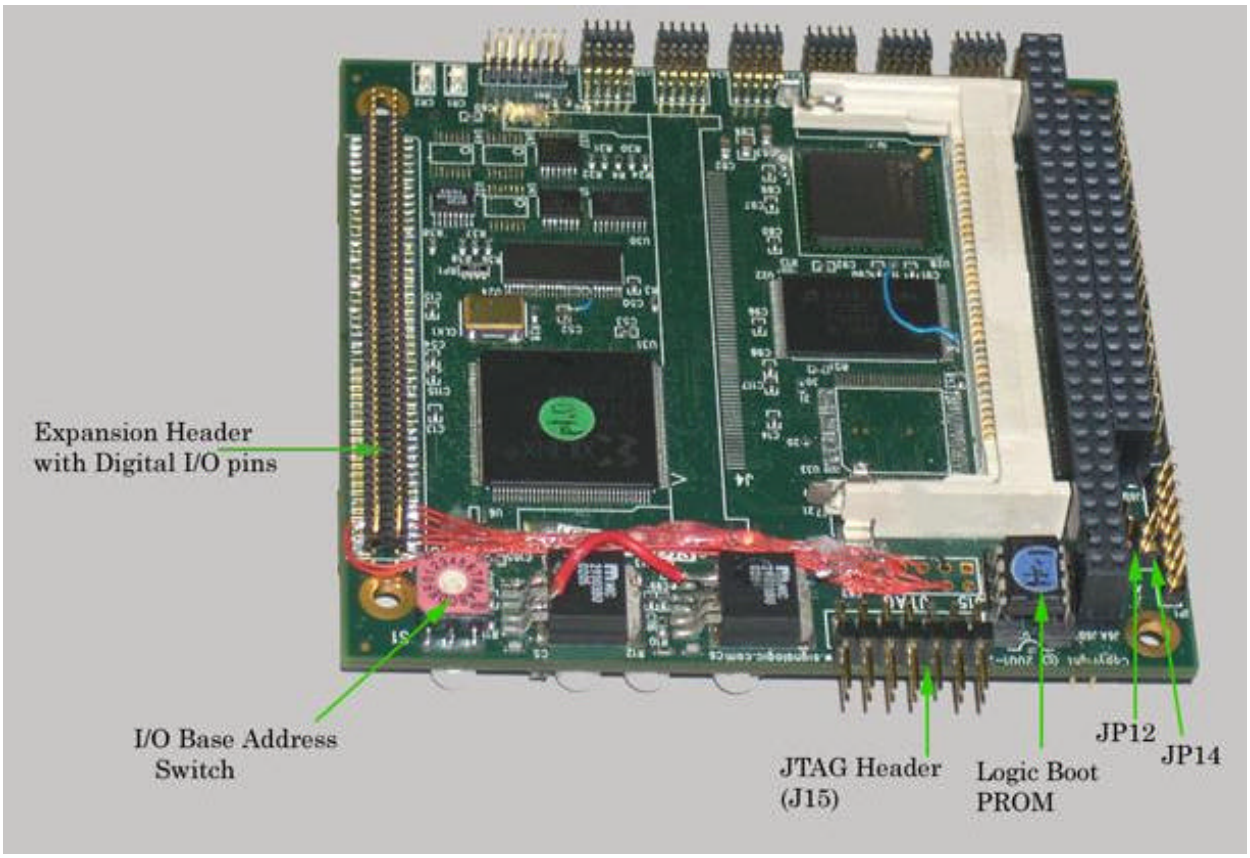


Figure 10-4, SigC6xxx-PC/104 Board with Expansion Header Installed

In the picture above, note the jumper wires installed to support digital I/O on a Rev C1 board. Rev D and higher boards do not require the wire modifications. The board in the picture is configured for a SigC67xx module (only one module site on top side), but the module is not installed in the picture.

For digital I/O pin assignment, see Figure 10-5 below.

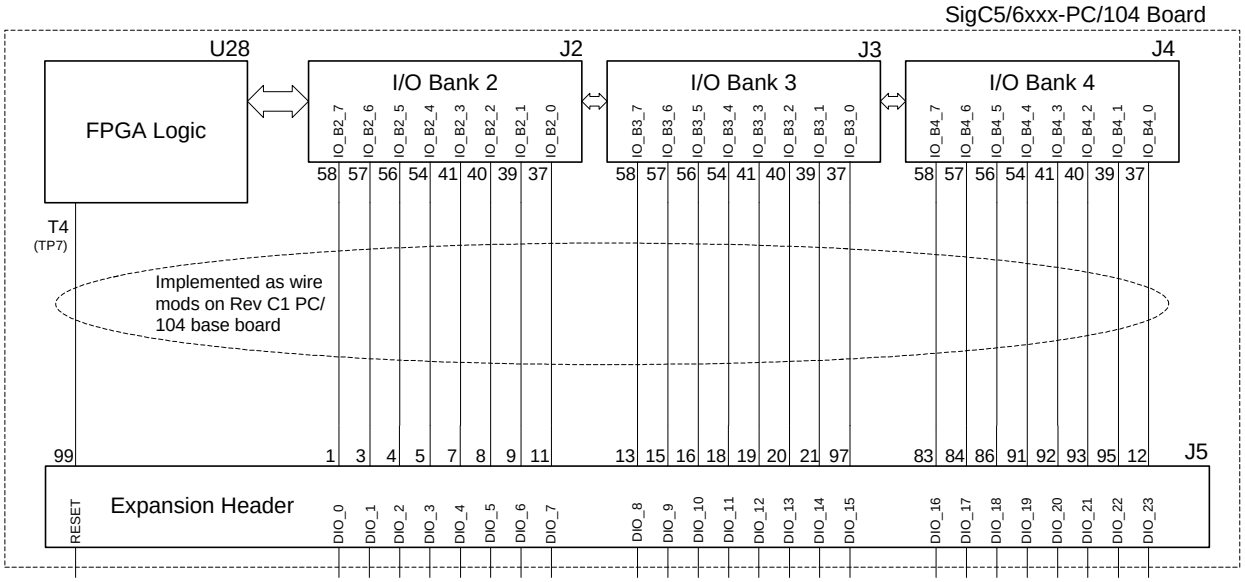

SigC5/6xxx-PC/104 Board
U28
J2
J3
J4
FPGA Logic
I/O Bank 2
I/O Bank 3
I/O Bank 4
IO_B2_7 IO_B2_6 IO_B2_5 IO_B2_4 IO_B2_3 IO_B2_2 IO_B2_1 IO_B2_0
IO_B3_7 IO_B3_6 IO_B3_5 IO_B3_4 IO_B3_3 IO_B3_2 IO_B3_1 IO_B3_0
IO_B4_7 IO_B4_6 IO_B4_5 IO_B4_4 IO_B4_3 IO_B4_2 IO_B4_1 IO_B4_0
58 57 56 54 41 40 39 37
58 57 56 54 41 40 39 37
58 57 56 54 41 40 39 37
T4
(TP7)
Implemented as wire mods on Rev C1 PC/ 104 base board
99
1 3 4 5 7 8 9 11
13 15 16 18 19 20 21 97
83 84 86 91 92 93 95 12
J5
RESET
Expansion Header
DIO_0 DIO_1 DIO_2 DIO_3 DIO_4 DIO_5 DIO_6 DIO_7
DIO_8 DIO_9 DIO_10 DIO_11 DIO_12 DIO_13 DIO_14 DIO_15
DIO_16 DIO_17 DIO_18 DIO_19 DIO_20 DIO_21 DIO_22 DIO_23

Figure 10-5, Digital I/O Pin Assignments

Each digital I/O bank is multiplexed with a PC104 board module site; banks correspond to module sites 1-to-1; i.e. digital I/O bank 2 is multiplexed with module site 2, etc. Typically, not all banks are active at one time – the most common situation being bank 1 is inactive due to module site 1 containing a processor module of some type. Note that the J-numbers shown in Figure 10-5 above correspond directly to the silkscreen markings on the board. For more information on physical module locations, see section 9.0 above, Module Interfaces.

Digital I/O banks may be enabled or disabled under software control. The default condition at power-up/system reset is all banks disabled. When any digital I/O bank is disabled, the physical digital I/O lines function only as dedicated module site signals.

The digital I/O peripheral contains two (2) types of registers, both accessible via host and DSP software:

- Direction
- I/O

Direction registers control whether individual I/O bits function as inputs or outputs; a zero specifies the bit functions as an input and a one specifies the bit functions as an output. At power-up/system reset, the default setting is all bits set to input (and all banks disabled, as mentioned above).

Digital I/O registers contain input bit values for each bit specified as input by its corresponding control register bit, or set external output bit values for each bit specified as output by its corresponding control register bit. Note that this arrangement makes digital I/O on SigC5xxx and SigC6xxx hardware fully bit-independent; i.e. bits do not need to be assigned as input or output in groups of 8 or 16 or other multiple.

 Copyright © Signalogic 2003-2005

Input bit values are constantly latched into holding registers that are "refreshed" at the master clock rate for onboard logic, typically ranging from 25 MHz to 40 MHz depending on module type and application requirements. Output bits are latched when written by software and retain their value until either cleared or PC104 board logic is reset or powered-off.

Both input and output bits retain their value when a bank is disabled; re-enabling a bank will restore the current values without need to for direction or I/O register update.

10.8.1 Digital I/O Testing

Below are the steps to run the digio_test.exe utility:

1) From command or console prompt, run digio_test.exe utility. Digio_test is a combined digital I/O test and EEPROM programming utility. At the initial start prompts, enter:

Entry	**Description**	**Comments**
2	Board Type	2 = PC/104 3 = PCI
2	Processor Type	2 = C5xxx 3 = C6xxx
2	Module Site	I/O bank for testing 2 = DIO bits 0-7 3 = DIO bits 8-15 4 = DIO bits 16-23 1 = DIO bits 24-31
1	Test Type	Digital I/O

A Test Type entry of 1 will cause all 8 bits of the specified bank to be treated as inputs. The screen print will continuously update the input result, so you should be seeing a display of 0xff without any external connections to J5. For example, to perform a test for I/O bank 2, take the following steps:

1) Clip a 10k resistor to J5-1.

2) Touch 10k R to ground (carefully).

3) Look at screen print results by the test program; you should see value change by one bit; for example if J5-1 is pulled down, then the value should change from 0xff to 0x7f.

4) Repeat. The procedure above can be repeated on the other 7 digital lines to verify that digital input is functioning as expected.

A Test Type entry of 2 will treat the first 4 bits of the specified I/O bank as input, and second 4 bits as output.

In making the above described measurements, please note that for Rev C1 board DIO lines are directly connected to board FPGA logic pins without series resistors (Rev D boards use 100 Ohm series resistors). ***Therefore, extreme care should be taken when handling DIO lines and connecting I/O devices to DIO lines. You should always use some type of series R (at least 100 ohm) when connecting pins to +Vcc and GND sources, to help avoid possible current surges and voltage spikes.***

Currently the digital I/O test utility cannot test more than one bank (8 bits) at a time.

10.8.2 Digital I/O Hardware Implementation Notes

Below are some notes about Digital I/O hardware implementation on SigC5xxx and SigC6xxx PC/104 boards:

1) Digital I/O inputs use FPGA logic with bi-directional buffers and a weak pull-up on each input. Direction control is bit-independent and selectable in either host or DSP software.

2) If you have a Rev C1 SigC5xxx or SigC6xxx PC/104 board, then some red or orange jumper wires should be installed on the board running to the expansion header (J5) to enable digital I/O. If full 24-bit digital I/O is enabled, then there should be jumper wires running from J5 to module sites J2, J3, and J4.

3) FPGA I/O pins are 5V tolerant, allowing digital I/O levels ranging from 3V to 5V to be used.

4) The maximum current that can be sinked or sourced from the FPGA I/O pins is approximately 24 ma for each pin. Therefore, when connecting I/O devices such as switches or LEDs, the total current for the device must be carefully considered. The recommendation is to ALWAYS use at least a 100 ohm series R to limit current and possible surges and spikes when switches are toggled. For example, for typical 5V LEDs, a 220 Ohm R is recommended.

10.8.3 Digital I/O Example C Code

The main.c (C5xxx and C6xxx SCI software) and melp_rt.c (MELPe deliverable) C source code files contain function calls to the SCI peripheral library to configure and access digital I/O on the PC/104 board expansion header (J5).

Main.c Digital I/O Examples

```
/* initial configuration */

        SetDigitalIOControl(RTC_IO_BANK2, 0x0f, DIGIO_ENABLE);  /* set upper 4 digital I/O bits
                                                                   to input, lower 4 bits to
                                                                   output */

     /* test digital I/O (inside real-time loop, for example) */
```

```
        dig_io_data = ReadDigitalIOData(RTC_IO_BANK2);

        WriteDigitalIOData(RTC_IO_BANK2, dig_io_data >> 4);  /* loop input bits to output */

/* clean-up, return to all-input configuration */

        SetDigitalIOControl(RTC_IO_BANK2, 0, DIGIO_DISABLE);  /* leave all bits set as input */
```

Melp_rt.c Digital I/O Examples

Melp_rt.c code contains C source examples to provide access to all panel controls and indicators used in an example enclosure. The configuration code is located near the start of the melp_real() function, and uses generic indicator and switch I/O examples, such as status LED, sync/activity LED, battery indicator, etc. Note that the example code is inside _DIGIO_BUILD_ conditional compiler statement, as follows:

```
#ifdef _DIGIO_BUILD_

/* initialize digital I/O (panel I/O for example user enclosure) */

/* bank 2 configuration */

   SetDigitalIOControl(RTC_IO_BANK2, 0x18, DIGIO_ENABLE);  /* tri-color status LED uses 2
                                                              output bits */

/* bank 3 configuration */

   SetDigitalIOControl(RTC_IO_BANK3, 0xf0, DIGIO_ENABLE);  /* dual tri-color sync/activity
                                                              LEDs use 4 output bits */

/* bank 4 configuration */

   SetDigitalIOControl(RTC_IO_BANK4, 0xe0, DIGIO_ENABLE);  /* battery indicator uses 3 output
                                                              bits */

/*
   Enable digital I/O and put a hold on enable state, to avoid Read/Write digital I/O calls
   below doing each time; note that this is not a problem unless other peripherals need to
   be accessed
*/

   SetDevice(RTC_DEV_DIGITAL_IO, RTC_DEV_ENABLE_HOLD | DEV_ENABLE);

#endif
```

In the example test code, the three (3) ReadDigitalIOData() and three (3) WriteDigitalIOData() function calls are located near end of the frame loop in melp_rt.c; look for the comment "update panel I/O". Also, note that this test code is within a conditional compiler statement, as shown below:

```
#ifdef _DIGIO_BUILD_

   bank2_io = ReadDigitalIOData(RTC_IO_BANK2) & 0xe7;  /* read inputs, mask output values */
   bank3_io = ReadDigitalIOData(RTC_IO_BANK3) & 0x0f;
   bank4_io = ReadDigitalIOData(RTC_IO_BANK4) & 0x1f;

   .
   .
   .
   process I/O -- modify bankn_io values as needed – set output bits

   note that output values do not need to be masked in ReadDigitalIO() calls if they
   will not be updated
```

 Copyright © Signalogic 2003-2005

```
      .
      .
      .

      WriteDigitalIOData(RTC_IO_BANK2, bank2_io);
      WriteDigitalIOData(RTC_IO_BANK3, bank3_io);
      WriteDigitalIOData(RTC_IO_BANK4, bank4_io);

   #endif
```

Note that the ReadDigitalIOData() and WriteDigitalIOData() function calls are made at frame (buffer) intervals, inside the real-time frame loop. There are three (3) calls to Read and Write each; each call handles 8-bits of digital I/O on the SigC5xxx-PC/104 expansion header. This corresponds to digital I/O banks 2-4 on the expansion header (note that bank 1 is not available because a DSP processor module is inserted in module site 1). Each bank is 8 bits; in the examples above, note that 23 out of 24 available bits are used.

Note also that the melp_rt example C source code includes commented-out printf lines that show how digital I/O bits are mapped; for example which I/O bits in which bank map to status or sync/activity LEDs, which bits map to a generic “Operating Mode” switch, etc. These source code lines were taken directly from C source code for the host test program used to test panel I/O items after enclosure assembly. Although left commented because CCS cannot handle print statements in a real-time system, this code should provide a definitive guide to the mapping and usage of all digital I/O bits in banks 2-4 in example units. Below is an excerpt from the "guidance code":

```
/*
      gotoxy(10, vofs);
      printf("Bit Packing = %d  ", io_bank2 & 7);
      gotoxy(30, vofs);
      printf("Operating Mode = %d  ", (io_bank2 & 0xe0) >> 5);
      gotoxy(50, vofs);
      printf("Audio Loopback = %d  ", io_bank3 & 1);
*/
```

 Copyright © Signalogic 2003-2005

10.9 Routing Logic

Note: on PC104 boards, the Routing Logic peripheral is a separate device. On PTMC boards, Routing Logic is co-located with host interface logic.

An overview of Routing Logic registers is given below. Note that these registers are only visible if the PERIPH_REG_EN bit in the GLOBAL_SETTINGS register and the TDM_RTSW_EN bit in the PERIPHERAL_CTRL register are set (see sections 10.3 and 10.4 above, Logic Registers and Peripheral Registers, for more information.

 Copyright © Signalogic 2003-2004

Routing Logic Registers

TDM Stream Switch and Control

Offset †	Routing Registers	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	
0	CLOCK_CTRL	CLK_GEN_EN ‡	FRAME_SYNC [1]		CLK_MODE [2]		CLK [3]			7-0
		RESERVED								15-8
1	CLOCK_PRD	CLK_PERIOD								15-0
2	STREAM_CTRL	RESERVED	C_CLK_SRC	B_CLK_SRC	A_CLK_SRC [4]	RESERVED	C_CLK_DIR	B_CLK_DIR	A_CLK_DIR [4]	
3	FLOW_CTRL	RESERVED	PTT_HW (RO)	PTT_HOST	CTS (RO)	CD (RO)	DSR (RO)	RTS	DTR	
4	PORT_CTRL	RSD_CLK_SRC	RSC_CLK_SRC	RSB_CLK_SRC	RSA_CLK_SRC [5]	RSD_CLK_DIR	RSC_CLK_DIR	RSB_CLK_DIR	RSA_CLK_DIR [5]	7-0
		RESERVED				PORTD_LEVEL	PORTC_LEVEL	PORTB_LEVEL	PORTA_LEVEL	15-8
5	T1E1_CTRL	RESERVED							TX_CLK_SEL	
6	RESERVED	REVISION MAJOR (RO) [6]				REVISION MINOR (RO) [6]				
7										
8	REV_ID	RS422_D_EN	RS422_C_EN	RS_422_B_EN	RS422_A_EN	MOD4_EN	MOD3_EN	MOD2_EN	MOD1_EN	7-0
9	A_TDM_SWITCH [7]	RESERVED			SERDES_REG_EN	RESERVED			T1E1_A_EN	15-8
0xa	B_TDM_SWITCH [7]	RS422_D_EN	RS422_C_EN	RS_422_B_EN	RS422_A_EN	MOD4_EN	MOD3_EN	MOD2_EN	MOD1_EN	7-0
0xb	C_TDM_SWITCH [7]	RESERVED			SERDES_REG_EN	RESERVED			T1E1_A_EN	15-8
0xc	SER_DES_RX [8]	RS422_D_EN	RS422_C_EN	RS_422_B_EN	RS422_A_EN	MOD4_EN	MOD3_EN	MOD2_EN	MOD1_EN	7-0
0xd	SER_DES_TX [8]	RESERVED			SERDES_REG_EN	RESERVED			T1E1_A_EN	15-8
0xe	RESERVED									
0xf	COMMAND	RESERVED						MODEM_MODE_EN	EXT_PTT_EN	

General Notes

† Routing Logic is a 16-bit peripheral; all registers are 16-bits deep although only first 8 bits of any register are currently defined. PERIPHERAL_REG_EN bit in GLOBAL_SETTINGS register and TDM_RTSW_EN bit in PERIPHERAL_CTRL register must be set (see Main Registers and Peripheral Registers documentation). Offset is relative to base I/O address (ISA bus) or BAR (PCI bus). All offsets assume 16 available locations.

‡ "_N" is used to denote an active low signal. "EN" is used as an abbreviation for "Enable".

All bits are read/write unless otherwise denoted by (RO) or (WO) designators. Registers with "STATUS" suffix contain one or more hardware inputs that may be read and monitored.

All register bits not defined are reserved and should be written as a zero. Undefined bits may not read as zero.

Register Specific Notes

[1] A non-zero FRAME_SYNC value enables frame sync generation when clock generation is enabled (CLK_GEN_EN bit is set). A FRAME_SYNC value of 01 enables 8-bit frame sync pulses, 10 enables 16-bit, and 11 enables 128-bit.

[2] CLK_MODE controls clock mode when internal logic clock generation is enabled (CLK_GEN_EN bit set). CLK_MODE values of 00 and 01 control clock output as shown in CLK_MODE and CLK Definition table at right. A CLK_MODE value of 10 uses CLK_PERIOD with CLK appended as upper 3 bits (19 bits total) to generate clock output. For example, with a 25 MHz input clock, CLK_MODE set to 10, and CLK+CLK_PERIOD set to 0x01000, generated clock output will be approx 6103 Hz.

[3] For CLK value definitions see table on this page. 25 MHz input clock is standard for C5xxx and C6713 boards; 37.5 MHz input clock is standard for C6711 boards.

[4] CLK_DIR bits define direction of clock and framesync signals from perspective of processor modules enabled by MODn_EN bits in X_TDM_SWITCH registers. 0 specifies an external clock; 1 an internal clock. External clocks may be driven by base board peripheral, peripheral module, or off-board external device in the case of RS-422 (CLK_SRC bit set to 0), or internal logic (CLK_SRC bit set to 1). Internal clock would typically be driven by the processor / DSP module. Note that if more than one MODn_EN bit is set in a X_TDM_SWITCH register, the lower bit value is assumed to be the "processor" module in determining clock and framesync direction.

[5] RSx_CLK_DIR and RSx_CLK_SRC bits function similarly to their STREAM_CTRL register counterparts, except external direction is defined as exterrnal to the board (input from external device), and internal is defined as either output from the stream (CLK_SRC set to 0) or from internal logic (CLK_SRC set to 1).

[6] REV_ID register value currently returns value 0x12, or v1.2. The REV_ID register is read-only.

[7] X_TDM_SWITCH registers allow board peripherals and modules to be connected to TDM streams. For C5xxx and C6xxx modules, TDM A stream is defined as McBSP0, TDM B stream is defined as McBSP1, and TDM C stream is defined as McBSP2 (note that some DSP device types do not have the third McBSP).

It is possible to have multiple sources connected to one TDM stream, as long as clock and framesync directions do not conflict. When multiple sources are defined, TX data lines are AND'ed together before connecting to the TDM stream. Note that clock and framesync direction control is determined by the STREAM_CTRL register.

X_TDM_SWITCH registers are 16-bit. Default values at power-on/reset are 0x000d for A_TDM_SWITCH, 0x0011 for B_TDM_SWITCH, and 0 for C_TDM_SWITCH. These default settings support a processor module in module site 1 and analog I/O module in site 3 or 4 connected to TDM stream A (McBSP0), and RS-422 port A connected to TDM stream B (McBSP1).

[8] SER_DES registers act as a peripheral and allow TDM streams to be accessed by host and DSP software, using peripheral register I/O or DMA controller transfers.

CLK_MODE and CLK Definition

CLK_MODE = 00 (CODEC Mode)

CLK	Output Rate (25 MHz Inp Clk)	Output Rate (37.5 MHz Inp Clk)
000	1200 Hz	1800 Hz
001	2000 Hz	3000 Hz
010	2400 Hz	3600 Hz
011	2700 Hz	4050 Hz
100	3200 Hz	4800 Hz
101	4000 Hz	6000 Hz
110	4800 Hz	7200 Hz
111	9600 Hz	14400 Hz

CLK_MODE = 01 (TDM Mode)

000	512 kHz	768 kHz
001	1.024 MHz	1.536 MHz
010	1.536 MHz	2.304 MHz
011	2.048 MHz	3.072 MHz
100	3.072 MHz	4.608 MHz
101	4.096 MHz	6.144 MHz
110	6.144 MHz	9.216 MHz
111	8.192 MHz	12.288 MHz

 Copyright © Signalogic 2003-2004

Routing Logic Register Notes

The PC104 or PTMC board routing logic peripheral can generate an internal clock. This is useful for applications when:

- specific bit rates are required
- an "independent clock" must be present that drives both processors /DSPs and also peripheral devices
- multiple devices or peripherals must be synchronized; for example, two RS-422 ports are mapped to two different McBSP ports on the DSP, but must be synchronized

To enable internal clock generation, the CLK_GEN_EN bit in the CLOCK_CTRL register should be set.

1. A non-zero FRAME_SYNC value enables frame sync generation when clock generation is enabled (CLK_GEN_EN bit is set). Below are possible FRAME_SYNC values:

FRAME_SYNC Value	Action (when CLK_GEN_EN is set)
00	None
01	Frame sync pulse every 8 bits
10	Frame sync pulse every 16 bits
11	Frame sync pulse every 128 bits

2. CLK_MODE controls clock mode when clock generation is enabled (CLK_GEN_EN bit is set). Below are possible CLK_MODE values:

CLK_MODE Value	Action (when CLK_GEN_EN is set)
00	Controls clock output as shown in CLK_MODE and CLK Definition table at right
01	Controls clock output as shown in CLK_MODE and CLK Definition table at right
10	Uses CLK_PERIOD value with CLK value appended as upper 3 bits (19 bits total) to generate clock output

11	Reserved

To give an example in the case where CLK_MODE is 10, with a 25 MHz input clock, CLK_MODE set to 10, and CLK+CLK_PERIOD set to 0x01000, generated clock output will be approximately 6103 Hz.

3. For CLK value definitions see table in Routing Logic Registers graphic above. A 25 MHz input clock is standard for C5xxx and C6713 boards; a 37.5 MHz input clock is standard for C6711 boards.

4. X_CLK_DIR bits define direction of clock and framesync signals from perspective of processor modules enabled by MODn_EN bits in X_TDM_SWITCH registers. 0 specifies an external clock; 1 an internal clock. External clocks may be driven by PC104 or PTMC board peripheral, peripheral module, or off-board external device in the case of RS-422 (X_CLK_SRC bit set to 0), or internal logic (X_CLK_SRC bit set to 1). Internal clock would typically be driven by the processor / DSP module. Note that if more than one MODn_EN bit is set in a X_TDM_SWITCH register, the lower bit value is assumed to be the "processor module" in determining clock and framesync direction, and other MODn_EN bits are interpreted as specifying a "peripheral module".

5. RSx_CLK_DIR and RSx_CLK_SRC bits function similarly to their STREAM_CTRL register counterparts, except external direction is defined as external to the board (input from external device), and internal is defined as either output from the stream (RSx_CLK_SRC set to 0) or from internal logic (RSx_CLK_SRC set to 1)

6. REV_ID register value currently returns value 0x21, or v2.1. The REV_ID register is read-only.

7. X_TDM_SWITCH registers allow board peripherals and modules to be connected to TDM streams. For C5xxx and C6xxx modules, TDM A stream is defined as McBSP0, TDM B stream is defined as McBSP1, and TDM C stream is defined as McBSP2 (note that some DSP device types do not have the third McBSP).

It is possible to have multiple sources connected to one TDM stream, as long as clock and framesync directions do not conflict. When multiple sources are defined, outputs from the stream are duplicated, and inputs to the processor module are AND'ed together. Note that clock and framesync direction control from the perspective of processor modules is determined by the STREAM_CTRL register.

X_TDM_SWITCH registers are 16-bit. Default values at power-on/reset are 0x000d for A_TDM_SWITCH, 0x0011 for B_TDM_SWITCH, and 0 for C_TDM_SWITCH. These default settings support a processor module in module site 1 and analog I/O module in site 3 or 4 connected to TDM stream A (McBSP0), and RS-422 port A connected to TDM stream B (McBSP1). The default values can be more clearly shown as follows (selected bit fields are shown in bold):

TDM STREAM A															
RSVD	RSVD	RSVD	SERDES	RSVD	RSVD	RSVD	T1E1	RS422-D	RS422_C	RS422-B	RS422-A	**MOD4**	**MOD3**	MOD2	**MOD1**
												1	**1**		**1**

In the above configuration, module site 1 is connected to module sites 3 and 4 on McBSP0. Clock and framesync direction are as defined by the A_CLK_DIR bit in the STREAM_CTRL register. Module site 1 can transmit data to both sites, and receive data from either site. Received data is logically AND'ed together – if only site 3 or site 4 is populated, then it is assumed that data, clock, and framesync lines float high for the unpopulated site.

TDM STREAM B															
RSVD	RSVD	RSVD	SERDES	RSVD	RSVD	RSVD	T1E1	RS422-D	RS422_C	RS422-B	**RS422-A**	MOD4	MOD3	MOD2	**MOD1**
											1	1	1		**1**

In the above configuration, module site 1 is connected to RS422-A port on McBSP1. Clock and framesync direction are as defined by the B_CLK_DIR and B_CLK_SRC bits in the STREAM_CTRL register.

8. SER_DES registers act as a peripheral and allow TDM streams to be accessed by host and DSP software, using peripheral register I/O or DMA controller transfers.

9. In the bit naming convention above, letters 'A', 'B', or 'C' used as a prefix indicate a TDM stream; used within a name or as a suffix indicate a port.

10.9.1 Routing Logic Example C Code

The main.c (C5xxx and C6xxx SCI software) and melp_rt.c (MELPe speech codec) example C source code files contain function calls to the SCI peripheral library to configure and access the Routing Logic peripheral on the PC/104 board.

Main.c Routing Logic Configuration Examples

Melp_rt.c Routing Logic Configuration Examples

The melp_rt.c C source code contains function calls to the SCI peripheral library to set Routing Logic registers on the PC/104 board, for example to change parameters such as bit rate and set clock direction. The example shown below enables on-logic clock and framesync generation, configures TDM A and TDM B streams, and sets clock direction and source for RS-422 port A.

The example code excerpt shown below is located near the start of void main(), within an if-statement that checks the fStandAloneMode flag, and within an `#ifdef _PERIPH_IO_BUILD_` statement (definition for which must be set in the Project Build Options), as shown below:

 Copyright © Signalogic 2003-2005

```
   #ifdef _PERIPH_IO_BUILD_

/* enable Routing Logic internal clock generation and set clock rate */

   clk_ctrl = ReadRoutingLogic(RTC_TDM_RTSW_CLOCK_CTRL);  /* read clock control register */

   if (sync_serial_enable == SS_PACKET_TRANSFER_MODE)

     clk_ctrl |= RTC_TDM_RTSW_CLK_GEN_EN | RTC_TDM_RTSW_FSYNC_8BIT;  /* enable logic clock,
                                                                        8-bit framesync */
   else

     clk_ctrl &= ~(RTC_TDM_RTSW_CLK_GEN_EN | RTC_TDM_RTSW_FSYNC);  /* disable logic clock,
                                                                      framesync */

/* internal clock generation setup */

   clk_ctrl &= ~RTC_TDM_RTSW_CLK;  /* first clear clock rate bits... */

   switch (lBaudRate) {            /* then select desired baud rate */

      case 1200:

         if (fUseBitPacking) clk_ctrl |= RTC_TDM_RTSW_CLK_1200;
         else clk_ctrl |= RTC_TDM_RTSW_CLK_2000;

         break;

      case 2000:

      /* this actually causes 1659 Hz rate until CLOCK_PRD register in Routing Logic is
         working */

         clk_ctrl |= RTC_TDM_RTSW_CLK_2000;
         break;

      case 2400:

         if (fUseBitPacking) clk_ctrl |= RTC_TDM_RTSW_CLK_2400;
         else clk_ctrl |= RTC_TDM_RTSW_CLK_3200;

         break;

      case 2700:
         clk_ctrl |= RTC_TDM_RTSW_CLK_2700;
         break;

      case 3200:
         clk_ctrl |= RTC_TDM_RTSW_CLK_3200;
         break;

      case 4000:
         clk_ctrl |= RTC_TDM_RTSW_CLK_4000;
         break;

      case 4800:
         clk_ctrl |= RTC_TDM_RTSW_CLK_4800;
         break;

      case 9600:
         clk_ctrl |= RTC_TDM_RTSW_CLK_9600;
         break;
   }

   WriteRoutingLogic(RTC_TDM_RTSW_CLOCK_CTRL, clk_ctrl);   /* write clock ctrl register */

/* RS-422 port A setup:

   -set clock/framesync direction and clock source

   -set framesync pulse count (none, 8, 16, or 128)
```

```
*/
/* set clock direction and source for TDM streams A and B */
   stream_ctrl = ReadRoutingLogic(RTC_TDM_RTSW_STREAM_CTRL);  /* read stream ctrl register */
/* set all clock directions to external (from DSP perspective) */
   stream_ctrl &= ~(RTC_TDM_RTSW_A_CLK_DIR | RTC_TDM_RTSW_A_CLK_DIR | RTC_TDM_RTSW_A_CLK_DIR);
   stream_ctrl |= RTC_TDM_RTSW_B_CLK_SRC;  /* set stream B clock source for internal clock */
   WriteRoutingLogic(RTC_TDM_RTSW_STREAM_CTRL, stream_ctrl);  /* write stream ctrl register */

/* set clock direction and source for RS-422 port A */
   port_ctrl = ReadRoutingLogic(RTC_TDM_RTSW_PORT_CTRL);   /* read port ctrl register */
   port_ctrl &= ~(RTC_TDM_RTSW_RSB_CLK_DIR | RTC_TDM_RTSW_RSC_CLK_DIR |
                  RTC_TDM_RTSW_RSD_CLK_DIR);  /* set ports B-D to external clock */
   port_ctrl |= RTC_TDM_RTSW_RSA_CLK_DIR;  /* set RS-422 port A clock direction to internal */
/* set RS-422 port A clock source to on-logic clock (applies when clock dir is internal) */
   port_ctrl |= RTC_TDM_RTSW_RSA_CLK_SRC;
   WriteRoutingLogic(RTC_TDM_RTSW_PORT_CTRL, port_ctrl);  /* write port ctrl register */

   #endif  /* _PERIPH_IO_BUILD_ */

}
```

 Copyright © Signalogic 2003-2005

10.10 T1/E1

Note: the T1/E1 peripheral is supported by PC104 boards only.

 Copyright © Signalogic 2003-2005

10.11 RS-422

Note: the multi-channel RS-422 peripheral is supported by PC104 boards only.

RS-422 ports are typically connected via PC104 board routing logic to one or more module site serial ports. For modules containing C5xxx or C6xxx DSP devices, serial ports directly translate to McBSP ports on the DSP devices. To enable connections, registers in the Routing Logic peripheral on the PC104 board must be initialized, either by host software or DSP software; for routing logic register information, see section 10.9 above, Routing Logic.

In addition to serial port connections, note that two additional RS-422 parameters are under software control:

- clock and framesync direction: whether clock and framesync are internal (driven by module site source) or external

- level control: bipolar signals for balanced operation (RS-422), and unipolar signals for RS-423 operation

RS-422 connectors on the SigC5xxx and SigC6xxx hardware are dual 5 row right-angle pin headers with 0.05" pitch between pins and 0.1" pitch between rows, and numbered J30-J33 on the board silkscreen (ports A – D). To see the physical layout of the RS-422 connectors, refer Figure 3-8 above, which also shows port A with a loopback plug installed that loops Tx data to Rx data for testing purposes.

Routing Logic Notes

See section 10.9 above, Routing Logic, for detailed information and notes about Routing Logic registers, including TDM stream configuration and RS-422 port settings.

RS-422 Serial Port Control Notes

1. Clock and framesync are available in pairs, one pair for each RS-422 port.

2. Clock direction and framesync direction cannot be controlled independently. If one is external, the other is also. This is a hardware limitation.

3. Depending on module type, not all RS-422 ports may be independent, or able to operate simultaneously. For example, with SigC54xx modules, two (2) serial ports (McBSP ports for C5xxx and C6xxx DSP devices) are available at the module edge. This means that two (2) RS-422 ports may be independently active at one time.

4. If more than one RS-422 port is mapped to the same serial port in the Routing Logic peripheral, then serial port outgoing Tx data will be transmitted over both ports (i.e. broadcast to more than one port), and incoming Rx data will be AND'ed together to the single serial port.

 Copyright © Signalogic 2003-2005

Below is a schematic excerpt of the SigC5/6xxx hardware RS-422 circuitry and external connectors:

 Copyright © Signalogic 2003-2005

10.12 USB 2.0

Note: the USB 2.0 peripheral is supported by PC104 boards only.

 Copyright © Signalogic 2003-2005

10.13 Ethernet 802.3

Note: the Ethernet 802.3 peripheral is supported by PC104 boards only.

 Copyright © Signalogic 2003-2005

10.14Ethernet 802.11b

Note: the Ethernet 802.11b peripheral is supported by PC104 boards only.

 Copyright © Signalogic 2003-2005

11 Component-Level Test and Measurement

11.1 Measuring Board Power Consumption

Below is a general procedure for measuring total 5V rail, 3.3V rail, and module site Vcc core voltage rail power consumption for SigC5xxx and SigC6xxx PC/104 boards up through Rev C1:

1) With the board removed from the PC or stand-alone enclosure, gently lift the power output pin (pin 4)of the 3.3V regulator (U18).

2) Solder on the free end of the lifted pin a 1 Ohm "brick" resistor, for example a large ceramic R obtainable at Radio Shack. Solder the other end of the brick resistor to the pad of pin 4.

3) Re-install the board and re-run test programs, Hypersignal or DirectDSP software, Code Composer Studio software, etc. as needed.

4) While the board is running, use a voltmeter or multimeter to measure the voltage drop across the 1 Ohm resistor. For a 1 ohm R, this value will also be the current measurement in A (Amperes). If you are using an R value other than 1 Ohm, then to obtain the current value, divide the voltage measurement by the R value.

5) Repeat steps 1-4 above to measure module site Vcc core voltage (U19). Note that SigC5xxx boards typically supply 1.6V output as module site Vcc core, and SigC6xxx boards typically provide 2.5V. However, note that also in many cases the DSP module performs additional regulation, for example to provide on-module DSP or logic PLD Vcc core voltages such as 1.8V and 1.6V.

6) To measure the incoming board 5V rail, steps 1-4 above may be repeated, except pin 1 (input) of the regulator should be used. Note that you should measure both regulators to get total 5V usage. If "power savings modifications" are installed on the board (see section 3.1.5 above, Power Consumption Reduction Board Modifications), it may be the case that the module site Vcc core regulator is using 3.3V from U18 rather than 5V.

Important Notes When Making Board Measurements

Please take all precautions in making the above measurements, including:

- Ensure that solder connections to lifted pins and/or pads are solid and firm
- Be careful when lifting regulator pins. Although these pins are big and relatively easy to work with, it is still possible to damage the pad for the pin, which could lead to problems in board reliability

 Copyright © Signalogic 2003-2005

Typical Power Measurements

Below is table giving typical PC/104 board power consumption measurements made in Signalogic labs.

Board or Module	Description	Approx Measured Power (W)
PC/104 Board	PC104 board logic, bus interface [1]	1.25
	RS-232	0.5
	RS-422	0.75
SigC67xx Module	C6711 processor running at 150 MHz	1.5
	4M x 32 SDRAM	1.2
	SigC67xx module logic	0.5
SigSD4 Analog I/O Module	4-channel analog module	0.6

Notes

[1] PC104 board power can be an additional 2W or more due to linear regulators on Rev C1 boards. For example, if a multiple processor module is installed, and draws heavily on 2.5V at the module edge, then additional power is lost in regulating from 5V to 2.5V on the PC104 board. See section 3.1.5 above, Power Consumption Reduction Board Modifications.